HOCHSCHULE
MITTWEIDA
UNIVERSITY OF
APPLIED SCIENCES

BACHELORARBEIT

Frau
Franziska Haffner

Automatisierte Erkennung von Malware auf Linux Systemen mit Hilfe von Indicators of Compromise

2017

Fakultät **Computer- und Biowissenschaften**

BACHELORARBEIT

Automatisierte Erkennung von Malware auf Linux Systemen mit Hilfe von Indicators of Compromise

Autorin:
Franziska Haffner

Studiengang:
Allgemeine und Digitale Forensik

Seminargruppe:
FO14w1-b

Erstprüfer:
Prof. Dr. rer. nat. Christian Hummert

Zweitprüfer:
Dr. Antje Winkler

Mittweida, August 2017

Bibliografische Angaben

Haffner, Franziska: Automatisierte Erkennung von Malware auf Linux Systemen mit Hilfe von Indicators of Compromise, 63 Seiten, 12 Abbildungen, Hochschule Mittweida, University of Applied Sciences, Fakultät Computer- und Biowissenschaften

Bachelorarbeit, 2017

Referat

Die Bachelorarbeit beschäftigt sich mit der Frage, inwiefern eine automatisierte Suche nach Malwarespuren mit Hilfe von „Indicators of Compromise“ auf Linux-Systemen realisierbar ist. Hierfür wird eine Liste von möglichen Malwareindikatoren in einer Linux Umgebung erstellt.

Durch den Autor wurden 25 Indikatoren aufgestellt. Zu jedem Indikator werden zwei Ansätze zur Automatisierung beschrieben. Ein Ansatz bezieht sich auf die automatisierte Datenerhebung. Der andere auf die automatisierte Analyse der erhobenen Daten. Für beide Ansätze werden innerhalb der Beschreibung Tools genannt, die für die Umsetzung in Frage kommen.

Um die entwickelten Ansätze zu testen, wurden ausgewählte Indikatoren in einem Skript umgesetzt. Durch das Skript wurde die Erkennung von Malware auf Grundlage von Veränderungen in der Userlandschaft, Code innerhalb des Kernels und der Prozesse getestet. Hierfür wurde der Code auf einer mit Malware infizierten virtuellen Maschine ausgeführt. Die anschließende Überprüfung der erhobenen Daten konnte jedoch keine der getesteten Malware Beispiele anhand von IoCs erkennen. Daher müssen für eine effektive Malware Erkennung weitere Indikatoren hinzugezogen werden.

Als Endergebnis der Arbeit konnte gezeigt werden, das eine teil-automatisierte Erkennung von Linux Malware mit Hilfe von IoCs möglich ist. Dabei kann vor allem die Datenerhebung automatisiert werden. Für die Analyse ist eine Automatisierung nur beschränkt möglich.

Abstract

The primary purpose of the study is to show that automated malware detection based on the use of indicators of compromise (IoC) is realizable for linux systems.

As one result of the thesis the author created an overview of 25 indicators that describe malicious behaviour within linux systems. For each indicator an aproach for an automated script implementation was described in a seperate paragraph.

The indicators for malicious behavior regarding manipulation of kernelcode, users and processes where tested in a test scenario with compromised virtual machines. Therefor a script has been created that realizes the automated part of data aquisition and analysis.

At the end of the test no malware sample could be detected by the choosen IoCs. This shows that automated malware detection needs more time in development to define high-quality IoCs.

Finally the thesis has shown that a semi-automated malware detection with IoCs is theoretical possible. The thesis has also shown the limits for the automated analysis.

Danksagung

An dieser Stelle möchte ich allen Menschen danken, die mich bei der Erstellung meiner Bachelorarbeit begleitet haben.

Ich danke meinen Betreuern, die sich immer wieder die Zeit nahmen, meine Arbeit zu kontrollieren und mir wertvolle Hinweise gaben, um nicht das wesentliche aus den Augen zu verlieren.
Ich danke allen Kollegen, die mir mit ihrem Fachwissen zur Seite standen und geduldig meine Fragen beantworteten.
Zudem danke ich meiner Familie, die sich nicht gescheut hat meine Arbeit unermüdlich auf Rechtschreibfehler zu kontrollieren.
Am Schluss möchte ich noch allen Freunden danken, die für mich da waren und mich durch ihre positive Art immer zum Weitermachen motiviert haben.

Vielen Dank.

I. Inhaltsverzeichnis

II. Abbildungsverzeichnis

III. Tabellenverzeichnis

Glossar

Backupfile Ein Backupfile ist eine Datei, die ein Abbild von wichtigen Daten darstellt. Sie ersetzt im Notfall die Originaldaten, wenn diese nicht mehr verfügbar sind, z.B. gelöscht wurden.

Blacklist Eine Blacklist beschreibt eine Liste von unerwünschten Zuständen. Innerhalb der Abwehr von Malware können so zum Beispiel IP-Adressen geblockt werden, wenn diese auf einer Blacklist stehen.

Brute-Force Bei Brute-force handelt es sich um eine Angriffstechnik, bei der Login und Passwörter durch Probieren von möglichen Kombinationen erraten werden.

Bytefolge Als Bytefolge wird ein Abschnitt, bestehend aus Maschinencode, bezeichnet.

ELF Als ELF wird das „Executable ans Linkable Format“ von Linux für ausführbare Dateien bezeichnet.

Executable Als eine Executable, wird eine ausführbare Datei bezeichnet.

Extension Als Extension wird die Dateiendung bezeichnet, die den Dateityp bzw. den Inhalt der Datei (z.B. „.ko“ bei Kernelmodulen beschreibt.

Hashwert Ein Hashwert ist das Ergebnis einer sogenannten Hashfunktion, d.h. einer mathematischen Funktion, um eine Prüfsumme zu berechnen. Hashwerte werden über Dateien berechnet, um diese eindeutig identifizieren zu können.

IoC IoC ist die Abkürzung für „Indicator of Compromise“. Damit werden Spuren bezeichnet, die auf eine Infizierung des Systems mit Malware hindeuten.

IoT Abkürzung für Internet of Things, Geräte die dauerhaft mit dem Internet vernetzt sind und selbstständig über dieses kommunizieren, um ihre Funktion zu erfüllen, z.B. IP-Kameras.

Malware Als Malware wird Software bezeichnet, die eine schädliche vom User nicht gewollte Funktion ausführt, wie z.B. das Verschlüsseln von wichtigen Dateien, ohne den Schlüssel preiszugeben.

Obfuskation Als Obfuskation wird eine Technik bezeichnet, bei der der Sourcecode eines Programms, durch erheblich schwerer zu analysierenden Code mit der gleichen Funktion, ersetzt wird.

RAM-Image Als RAM Image wird ein Abbild des Hauptspeichers bezeichnet. Mit diesem Abbild kann zu einem späteren Zeitpunkt der Zustand des Systems zum Zeitpunkt der Sicherung untersucht werden.

Signatur Im Kontext von Malware ist eine Sigantur ein Abschnitt einer schädlichen Datei, der diese eindeutig Identifiziert.

Snapshot Als Snapshot wird das Abbild einer virtuellen Maschine im laufenden Zustand beschrieben. Zu einer virtuellen Maschine können mehrere Snapshots gespeichert werden und somit zwischen verschiedenen Zuständen des Systems gewechselt werden.

Triage Image Als Triage Image wird ein Teil Abbild eines Systems bezeichnet, aus dem die wichtigsten Informationen für eine Analyse des System gewonnen werden können.

virtuelle Maschine Als virtuelle Maschine (oder kurz VM) wird ein emuliertes Host-System verstanden, welches die Hardware des real existierenden Systems nutzt und den Rest des Betriebssystems durch Software umsetzt.

Whitelist Eine Whitelist beschreibt eine Liste von erwünschten Zuständen. Bei der Abwehr von Malware können Listen mit erwünschten IP-Adressen erstellt werden, um Verbindungen nur zu den gelisteten Adressen zuzulassen und anderen Netzwerkverkehr zu blockieren.

1 Einleitung

1.1 Motivation

Malware ist durch ihre rasante Entwicklung in den letzten Jahren zu einer Bedrohung herangewachsen. Laut Ralf Benzmüller vom G Data Security Blog entstand 2016 „alle 4,6 Sekunden ein neuer Schadprogrammtyp“ [Ben17]. Noch wird 99% aller Malware für Windows geschrieben [Ben17]. Die große Verbreitung des Betriebssystems macht es zum Ziel von organisierter Kriminalität im Bereich Cybercrime. Malware dient den Angreifern dabei als Werkzeug, um sich illegal Zugang zu Systemen zu verschaffen, Daten zu stehlen oder bleibende Schäden zu verursachen.

Linux hingegen wird von vielen Experten als sehr sicher eingeschätzt - es ist keine große Verbreitung von Malware über mehrere Linux-Systeme bekannt, wie es vergleichbar bei Windows passiert. Diese Feststellung verleitet dazu, Linux-Systeme nicht aktuell zu halten und Security-Standards zu vernachlässigen. In Folge dessen stößt Linux-Malware beim Ausnutzen von Sicherheitslücken kaum auf Widerstände [DV16].

Dies geschieht in einer Zeit, in der Linux immer mehr an Bedeutung gewinnt. Das am weitesten verbreitetste Betriebssystem für mobile Geräte ist Android und basiert auf einem Linux-Kernel (siehe Abb. 1.1). Im Bereich der Cloud- und Webserver setzt sich das freie Linux ebenfalls gegenüber dem kostenpflichtigen Windows durch. 66,6% aller Webserver laufen auf einem Unix oder Linux System. Davon die meisten mit 35,9% auf einer Ubuntu-Distribution [w3t17].

Eine aktuelle Meldung des BSI zeigt, dass über 20.000 deutsche Systeme Software-Updates für Nextcloud und ownCloud benötigen, Software die für das Verwalten von Dateien auf Cloudservern zuständig ist. Die Server enthalten sensible Daten, wie Kontaktlisten, wichtige Dateien und Emails und ergeben somit attraktive Ziele für einen Angriff [Eik17]. Wer solche Sicherheitslücken ignoriert, kann selbst auf Linux zum Opfer von Malware werden.

Auch wenn das eigene System aktuell und sicher ist, kommt es bei wachsenden Unternehmen oft vor, dass fremde Infrastruktur in die bestehende Infrastruktur integriert werden soll. Hierbei entsteht das Risiko, infizierte Systeme in das eigene Netzwerk zu übernehmen. Vor einer solchen Fusionierung sind alle zu integrierenden Systeme auf Malware zu untersuchen.

Die Firma T-Systems Multimedia Solutions GmbH bietet ihren Kunden eine Überprüfung von Windows Systemen auf Malware an. Meist handelt es sich um Server, die möglichst während des Betriebes ohne Unterbrechung überprüft werden sollen. Für das Scan-

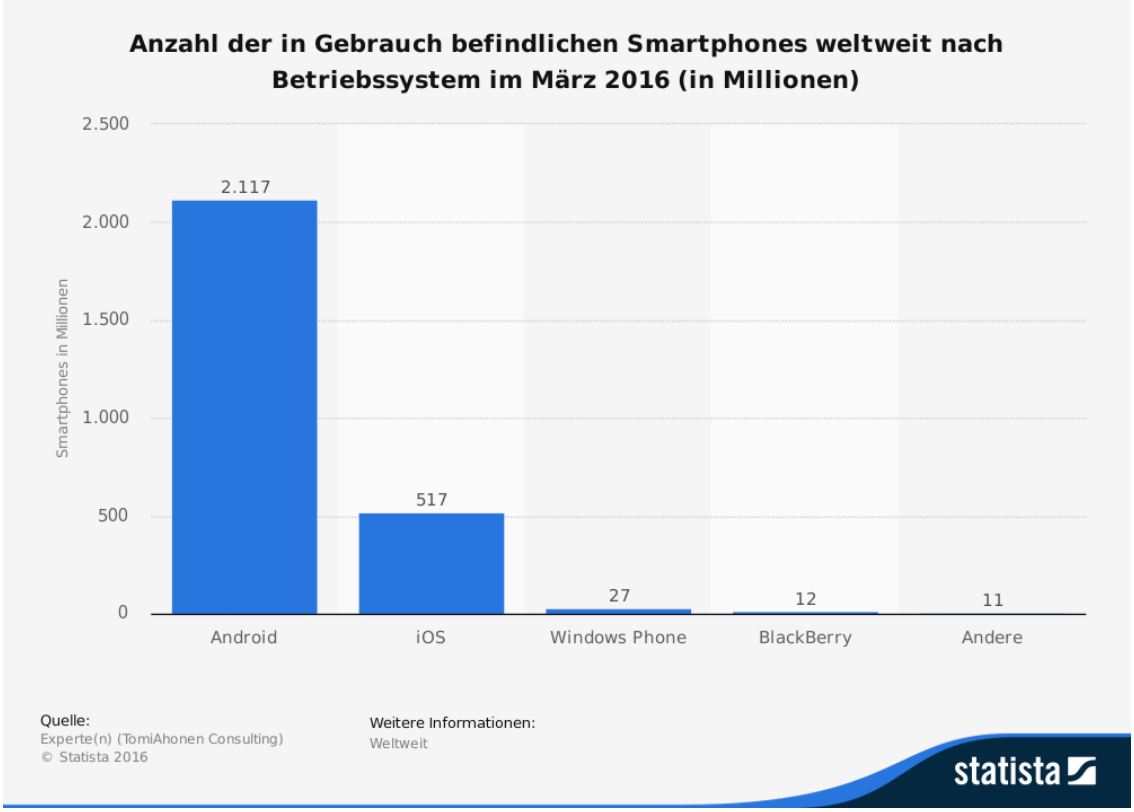

Abbildung 1.1: Anzahl der im Gebrauch befindlichen Smartphones nach Betriebssystem. Stand Mai 2016. [Tho16]

nen solcher Systeme eignet sich ein automatisches Skript, welches selbstständig Daten sammelt und anschließend eine Auswertung vornimmt. Dies minimiert zudem den personellen Aufwand, da meist nur eine Nachkontrolle der Daten erfolgen muss. Zudem können in kurzer Zeit viele Systeme überprüft werden, da eine intensive forensische Analyse sehr viel Zeit in Anspruch nehmen würde.

Die Erkennung von Malware auf den Systemen basiert auf der Suche nach „Indicators of Compromise“ (IoC), also nach Spuren, die Malware auf einem System hinterlässt. Eine genaue Erläuterung von IoCs findet sich unter Abschnitt 2.4.

1.2 Aktueller Forschungsstand

Der folgende Abschnitt soll einen Einblick in aktuelle Forschungsansätze im Bereich der automatisierten Malware Erkennung und Analyse geben.

Viele kommerzielle Malware Scanner basieren auf statischen Ansätzen der Malware Analyse. Die Arbeit von Moser [MKK07] zeigt jedoch auf, dass bereits einfache Verschleierungstechniken, wie Quellcodetransformation genügen, um Malware vor dem Scanner zu verstecken. Daher gibt es viele wissenschaftliche Arbeiten, die sich mit der dy-

namischen Erkennung von Malware auseinander setzen. Diese Methode ist wesentlich robuster gegenüber Veränderungen des Schadcodes, da der Fokus bei der Erkennung der Malware auf bestimmten Verhaltensmustern liegt.

In der Arbeit von Damri und Vidyarthi [DV16] werden verschiedene Ansätze für die automatische dynamische Malware Erkennung vorgestellt. Als wichtigste Ansätze befinden die Autoren:

- **Systemcall-basierter Ansatz:** Systemcalls dienen der Interaktion von Prozessen mit dem Kernel, um bestimmte, vom Betriebssystem bereitgestellte Funktionen, auszuführen, wie zum Beispiel das Öffnen einer Datei. Zur Erkennung von Malware werden die von Prozessen verwendeten Systemcalls überwacht. Malware verwendet bestimmte Systemcalls häufiger, daraus können Muster abgeleitet werden.
- **Kernel-basierter Ansatz:** Durch die Überwachung des /proc Dateisystems können Veränderungen im Kernel zu Laufzeit festgestellt werden. Proc enthält dabei Informationen über Hardwarekonfigurationen, angeschlossene Geräte, den Speicher und über laufende Prozesse. Hierfür existiert zu jedem Prozess ein Unterverzeichnis, benannt nach der Prozess ID.
- **Prozesstabellen-basierter Ansatz:** Jeder aktive Prozess erzeugt im Kernel ein sogenanntes task_struct Element (oder auch Eintrag in der Prozesstabelle), welches sämtliche Informationen über den Prozess enthält. Daraus kann das Laufzeitverhalten des Prozesses abgeleitet werden. Im direkten Vergleich von schädlichen und harmlosen Prozessen können Muster aus dem Verhalten abgeleitet werden. Daraus kann später entschieden werden, ob ein unbekannter Prozess eine schädliche Wirkung besitzt.
- **ELF-Struktur-basierter Ansatz:** Das „executable and linkable“ Format (ELF) ist das Linux-spezifische Dateiformat für ausführbare Dateien. Die Felder im Header können zwischen harmlosen Dateien und schädlichen Dateien verglichen werden, um Muster abzuleiten und unbekannte Executables richtig einzuordnen.
- **Hybrider Ansatz:** Hybride Ansätze kombinieren zwei oder mehr der vorhergehenden Methoden, um Malware zu erkennen.

Die Autoren erkannten zudem das Problem, dass Systemcall basierte Ansätze ebenfalls durch Obfuskationstechniken[1] von Malware zu schlechten Ergebnissen führen. Verschleierung von schädlichem Code kann also selbst bei dynamischen Ansätzen effektiv die Erkennung von Malware verhindern.

Die Arbeit von Burguera et al. [BZNT11] setzt bewusst auf den Systemcall-basierten Ansatz für die Entwicklung ihrer Applikation „Crowdroid“, die schädliche Apps auf Android Geräten erkennt. Obwohl andere Ansätze eine tiefere Analyse und somit detailliertere und sicherere Ergebnisse liefern, halten die Autoren an dem Ansatz fest. Hauptgrund ist

[1] Verschleierungstechnik, die die Auswertung von Sourcecode erschwert, z.B. durch das Einfügen von funktionslosen Code

der geringe Speicher und Leistung, die bei mobilen Geräten zur Verfügung stehen. Die Überwachung von Systemcalls ist im Vergleich die Ressourcen sparsamste Methode.

Dahingegen liefert der ELF-Struktur-basierte Ansatz in der Arbeit von Shazad und Farooq [SF12] eine Erkennungsrate bei Malware von über 99%. Grundlage hierfür ist das ELF-Dateiformat für Executables unter Linux. Die Autoren extrahieren Informationen aus den Feldern im Header und vergleichen diese zwischen harmlosen und schädlichen Dateien. Aus dem Vergleich können sie Felder bestimmen, die eine besonders hohe Aussagekraft hinsichtlich der Schädlichkeit von Executables besitzen. Als Ergebnis der Arbeit stellen die Forscher den „ELF-Miner“ vor, einen Algorithmus der automatisch ELF-Dateien als schädlich oder harmlos klassifizieren kann und dabei mit einer geringen falsch-positiv Rate arbeitet. Allerdings genügt die Betrachtung des ELF-Headers nicht, um sämtliche Malware aufzudecken, da sich nicht immer Malware-Executabels auf dem System befinden oder leicht gefunden werden können.

Einen interessanten Ansatz zur automatischen Extraktion von Indicators of Compromise liefert die Arbeit von Liao et al. [LYW+16]. Die Autoren kombinieren verschiedene Natural Language Processing (NLP) Ansätze mit Regular Expressions (RE), um IoCs aus Texten von technischen Reports in das OpenIOC Format zu übertragen. Das OpenIOC Format ist hierbei ein von Mandiant[2] entwickeltes XML-Format, um einen Standard für die Dokumentation von IoCs zu etablieren und gleichzeitig ein maschinenlesbares Format zu haben, von dem aus IoCs in andere Formate, wie Yara-Rules, kompiliert werden können.

Die Dokumentation von IoCs im OpenIOC Format bietet, aufgrund der Maschinenlesbarkeit, eine gute Grundlage für eine spätere automatisierte Überprüfung von Systemen auf Malware.

Bei den oben aufgeführten Papern handelt es sich jeweils um Ansätze, die auf maschinellem Lernen basieren. Ein idealer Ansatz wäre sicherlich, die gelernten Muster der Algorithmen als IoCs im OpenIOC Format abzubilden. Dieser Ansatz ist jedoch sehr aufwendig, da ein Lernalgorithmus, bis er qualitativ hochwertige Ergebnisse liefert, sehr viele Testdatensätze und Anpassungen benötigt. Zudem sind IoCs sehr vielfältig, dh. sie können sich auf unterschiedliche Datenstrukturen beziehen, so dass für jede Kategorie von IoC ein eigener Lernalgorithmus benötigt werden würde.

Dies ist im Rahmen einer Bachelorarbeit nicht möglich. Daher ist es notwendig schon bekannte IoCs zu sammeln und übersichtlich aufzuführen. Aufgrund der unterschiedlichen Konventionen zur Dokumentation und der fehlenden Zusammenarbeit von Forschungseinrichtungen existiert eine solche Übersicht für Linux Malware derzeit nicht.

[2] Mandiant ist ein IT-Sicherheitsunternehmen, das 2013 von FireEye übernommen wurde [fir17]

1.3 Aufgabenstellung

Diese Bachelorarbeit beschäftigt sich mit der Frage, ob eine automatisierte Suche nach IoCs auf Linux System äquivalent zu Windows möglich ist.

Hierbei wird auf eine Betrachtung von Linux-basierten Betriebssystemen für mobile Geräte verzichtet. Der Fokus liegt auf Linux-Distributionen, die hauptsächlich im Server-Bereich eingesetzt werden, da Linux im Vergleich zu Desktop-Systemen, hier weiter verbreitet ist.

Als Endergebnis der Arbeit soll eine Übersicht mit IoCs für Linux erstellt werden. Für jedes IoC erfolgt eine genaue Beschreibung und die Entscheidung, ob eine automatisierte Erhebung und Auswertung möglich ist. Ist diese möglich, wird der Ansatz näher beschrieben.

1.4 Aufbau der Arbeit

In Kapitel 2 sollen die theoretischen Grundlagen für das Verständnis der Arbeit gelegt werden. Hierzu zählt eine nähere Betrachtung der Themen Malware Analyse und Indicators of Compromise, sowie Malwareverhalten speziell innerhalb von Linux Systemen.

Im Hauptteil der Arbeit wird das Vorgehen zur Erstellung einer Übersicht von IoCs und die Überprüfung auf Automatisierungsmöglichkeiten vorgestellt (siehe Kapitel 3). Anschließend folgen in Kapitel 4 die Ergebnisse in Form einer tabellarischen Übersicht und ein Test von IoCs für ein konkretes Malwarebeispiel in einer virtuellen Umgebung.

Unter Kapitel 5 erfolgt die kritische Bewertung der eigenen Arbeit und ein Vergleich, mit der Umsetzung der Malware Erkenung bei Windows. Es wird ein Fazit in Hinsicht auf die Erfüllung der Zielsetzung gezogen. Zusätzlich gibt der Autor einen Ausblick auf zukünftige Arbeiten für die diese Bachelorarbeit als Grundlage dienen kann.

2 Theoretische Grundlagen

Das folgende Kapitel dient als Überblick für die verschiedenen Themen die innerhalb der Arbeit betrachtet wurden. Es soll die theoretischen Grundlagen für das Verständnis der Arbeit legen. Dabei wird zuerst ein Überblick über das Betriebssystem Linux gegeben. Anschließend wird das Thema Malware und die Analyse von Malware erklärt. Weitere speziellere Themen sind die Abschnitte zu den IoCs und dem Verhalten von Malware auf Linux. Das Kapitel schließt mit der Vorstellung der bereits existierenden Windows Lösung für die Erkennung von Malware.

2.1 Linux

Bei Linux handelt es sich um ein Betriebssystem, welches auf dem Linux Kernel aufbaut. Erst mit zusätzlichen Softwarepaketen ergibt sich, zusammen mit dem Kernel, ein vollwertiges Betriebssystem. Von Linux existieren unterschiedliche Ausführungen, sogenannte Distributionen, die sich in den verwendeten Softwarepaketen unterscheiden. Der Kernel dient dabei als Schnittstelle zwischen der Software und der Hardware (siehe Abb.: 2.1). Programme können über Systemaufrufe (Systemcalls) Anfragen an den Kernel senden, die dieser an die Hardware weiterleitet. Der Kernel kommuniziert über sogenannte Gerätetreiber mit der Hardware [Pol16].

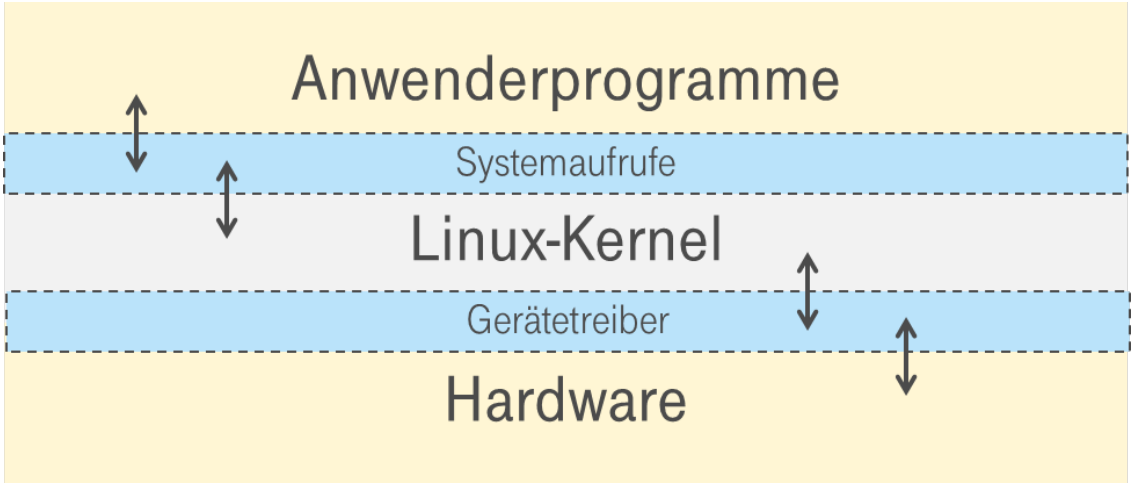

Abbildung 2.1: Kommunikation zwischen Programmen, Kernel und Hardware (Quelle: nach [Pol16])

Über die Shell kann ein Benutzer Systemaufrufe direkt an den Kernel weiterleiten. Linux ist ein multiuser Betriebssystem, dh. mehrere User können an mehreren Terminals arbeiten. Jeder User wird über eine eindeutige ID (UID) identifiziert. Die UID 0 ist dabei für den root User reserviert. Jeder User ist zudem Mitglied in mindestens einer Gruppe.

Jede Gruppe besitzt ebenfalls eine ID (GID). Zusätzlich zu den normalen Usern existieren Systemnutzer, die keine Login Shell benötigen, d.h. sich nicht wie ein normaler User einloggen. Sie werden von Programmen benötigt, um auf Dateien zugreifen zu können [Pol16].

2.1.1 Grundkonzepte Linux

Im folgenden sollen bestimmte Grundkonzepte, die für das weitere Verständnis der Arbeit relevant sind, vorgestellt werden [Pol16].

Dateisystem

Unter einem **Dateisystem** versteht man eine Organisation für die Ablage von Dateien auf einem Datenträger. Unter Linux wird häufig auf das „Extended File System“ (ext) zurückgegriffen. In vielen Distributionen hat es sich als Standard etabliert. Mit der Version „ext3“ wurde die Journal-Funktion hinzugefügt, welche die Änderungen an Daten protokolliert und somit die Wiederherstellung nach Systemabstürzen sicher stellt. Die aktuelle Version ist „ext4“.

Dateien

Sämtliche Ressourcen in Linux werden wie **Dateien** behandelt, d.h. auch für Hardware existieren Dateien, über die der Zugriff realisiert wird. Auch Linux unterscheidet in sichtbare und für User versteckte Dateien. Versteckte Dateien beginnen mit einem Punkt im Namen und werden bei einer normalen Anzeige der Dateien und Verzeichnisse nicht aufgelistet.

Jede Datei verfügt zudem über vielfältige Attribute, wie einen Besitzer und eine Gruppenzugehörigkeit. Dieser legt fest, welche Berechtigungen andere Nutzer für die Datei haben. Es können Rechte zum Lesen, Schreiben oder Ausführen vergeben werden. Der Trenner für Pfadangaben ist „/“ (Slash), anstatt des „\“ (Backslash) der unter Windows verwendet wird.

Für die Verwaltung der User wichtige Dateien:

- **/etc/passwd**: listet alle User (inklusive Systembenutzer) mit zugehöriger UID und GID auf, führt die vom User zu benutzende Login-Shell auf
- **/etc/group**: listet alle im System existierenden Gruppen mit dazugehörigen Usern auf
- **/etc/shadow**: führt alle User mit dazugehörigem Passwort-Hash auf

Verzeichnisbaum

Die wichtigsten **Verzeichnisse** werden in den verschiedenen Distributionen gleich benannt, da sich die meisten am Filesystem Hierarchy Standard (siehe Abb.: 2.2) orientieren. Dieser Standard wurde für unixoide Betriebssysteme entwickelt und legt die Verzeichnisstruktur für diese fest. Geringe Abweichungen können unter den einzelnen Distributionen vorkommen.

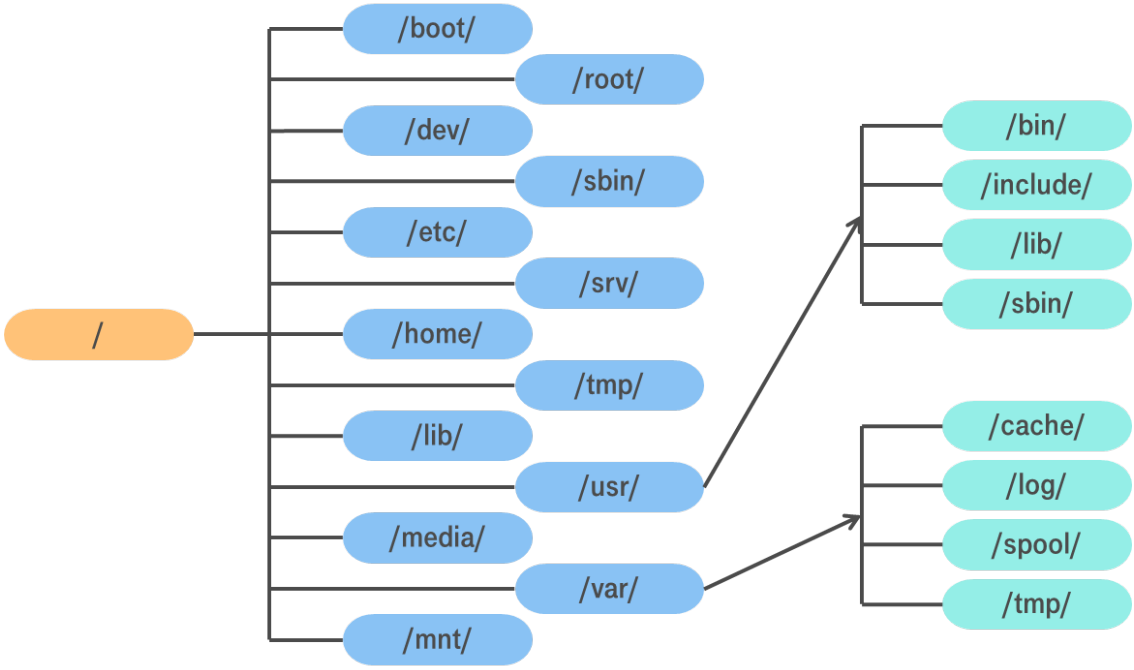

Abbildung 2.2: Filesystem Hirachie Standard (nach Quelle: [nep16])

Alle Verzeichnisse stammen vom Wurzelverzeichnis „/“ ab. Unix besitzt eine Wurzelpartition, die anderen Partitionen werden an beliebiger Stelle im Verzeichnisbaum „angehangen“ (mount).

Es folgt eine Beschreibung der wichtigsten Verzeichnisse:

- **/bin:** enthält die wichtigsten Binaries (binäre Programme) für die Ausführung des Systems, können von allen Usern ausgeführt werden
- **/boot:** Ablageort des Kernels und aller Dateien, die zum Starten benötigt werden
- **/dev:** Gerätedateien (Schnittstellen zum Kernel), zum Ansprechen der Hardware
- **/etc:** System-weite Konfigurationsdateien, Dateien für Benutzerverwaltung
- **/home:** für jeden User existiert ein gleichnamiges Verzeichnis im Home-Verzeichnis, normale User haben uneingeschränkte Rechte im eigenen Home-Verzeichnis und meist nur lesenden Zugriff auf alle anderen Verzeichnisse
- **/lib:** Systembibliotheken, Kernelmodule

- **/mnt und /media:** dienen zum Einbinden von gemounteten Laufwerken und Geräten
- **/proc:** enthält Kernelinformationen, z.B. Unterverzeichnisse zu allen aktiven Prozessen
- **/root:** Homeverzeichnis des root Users
- **/sbin:** System-Binaries, dürfen nur von Usern mit administrativen Rechten ausgeführt werden
- **/tmp:** temporäre Dateien, wird normalerweise beim Systemstart gelöscht
- **/usr:** „Unix System Ressources“, Unterverzeichnisse z.B. */usr/src* enthält Quelltexte aller Programme für das Standardsystem
- **/var:** Dateien, die von Programmen häufig geändert werden (Zwischenlagerung von Mailserver Mails in */var/spool* und Speicherung der Logs in */var/log*)

Paketverwaltung

Unter Linux werden Programme in Pakete unterteilt, die einzelne Funktionen übernehmen. Diese Pakete werden über einen Paketmanager verwaltet. Dieser übernimmt die Installation der Pakete und löst gleichzeitig alle Abhängigkeiten der Pakete untereinander auf. So werden bei der Installation eines Paketes über einen Paketmanger gleichzeitig alle für das Paket benötigten Pakete nachinstalliert.

Die verschiedenen Distributionen haben standardmäßig unterschiedliche Paketmanager installiert. Unter Debian und Ubuntu wird auf „Advanced Packaging Tool“ (APT) zurück gegriffen. RedHat verwendet den RPM Paketmanager zur Verwaltung der Pakete.

2.2 Malware

Als Malware wird Software bezeichnet, die entwickelt wurde, um auf Computersystemen unrechtmäßig Schaden zuzufügen [Asc14]. Die Entwickler von Malwarecode verfolgen dabei verschiedene Ziele. Oft erzielen sie einen finanziellen Vorteil, über die Erpressung von Opfern oder den Verkauf von kriminellen Dienstleistungen, die über die Malware bereitgestellt werden. Nicht nur private Nutzer von Heimcomputern sind dabei potenzielle Ziele, auch große Unternehmen sind attraktiv.

2.2.1 Arten von Malware

Es existiert eine Vielzahl von Malware, deren Code immer wieder modifiziert wird, um nicht von Antiviren-Systemen erkannt zu werden. Malware unterliegt so einer großen Vielfalt. Es gibt unterschiedliche Ansätze, eine Einteilung der bisher bekannten Schadsoftware vorzunehmen. Beispielhaft kann in Art der Verbreitung und Schadwirkung unterschieden werden.

Klassifizierung nach Verbreitungsstrategie

Die hier aufgeführten Arten von Malware werden nach Art ihrer Verbreitung unterschieden. Hierbei wird unterschieden in eine selbstständige Verbreitung und eine von dem User initiierte Verbreitung, durch z.B. das Ausführen einer schädlichen Datei [Asc14] [Hum17, S. 201 f.].

- **Viren:** Viren sind schädliche Programme, die sich selbstständig verbreiten, d.h. ohne manuell durch den User gestartet zu werden. Sie infizieren andere Programme, die sogenannten Wirte und machen diese meist unbrauchbar.
- **Würmer:** Sie ähneln in ihrer Arbeitsweise stark den Viren, befallen jedoch weniger Programme sondern vielmehr Hardware, also USB-Speichermedien oder Festplatten. Ein Wurm verbreitet sich nach der Ausführung selbst weiter. Im Gegensatz zum Virus verändert er keine fremden Dateien oder Bootsektoren.
- **Rabbit:** Der Rabbit ist ein Ausnahmefall des Wurms. Die Verbreitung des Schadcodes erfolgt ebenfalls selbstständig. Er vervielfältigt sich jedoch nicht, sondern legt immer genau eine Kopie von an und löscht sich an der ursprünglichen Stelle.
- **Trojaner:** Trojaner sind schädliche Programme, die sich nicht selbstständig verbreiten können. Der Code wird meist in vermeintlich nützlichen Programmen versteckt. Wenn der User das Programm ausführt, führt er gleichzeitig den Trojaner aus, welcher heimlich im Hintergrund ein anderes Programm installiert. Sie werden meist über Email-Anhänge verbreitet.

Klassifizierung nach Schadwirkung

Eine weitere Möglichkeit Malware zu Klassifizieren bietet sich in der Einteilung nach Schadwirkung, also ihrer Funktion und Auswirkung auf das betroffene System [Asc14] [Hum17, S. 201 f.].

- **Logic Bomb:** Bei der Logic Bomb handelt es sich um ein Programm, das zunächst keinen Nutzen hat. Erst wenn eine bestimmt Bedingung erfüllt ist, kommt die schädliche Funktion zum Vorschein.
- **Backdoor:** Eine Backdoor hinterlässt auf einem infizierten System eine Schwachstelle, über die der Angreifer immer wieder, unter Umgehung der Zugriffskontrolle, Zutritt in das System erhält. Ein Spezialfall ist das sogenannte **Rootkit**, das gleichzeitig auch administrative Rechte für den Angreifer garantiert.
- **Bacteria:** Ziel dieses Programms ist die Auslastung von Systemressourcen, indem sich das Programm oft vervielfältigt und selbst ausführt. Somit verbraucht es z.B. Speicherplatz auf der Festplatte oder im Hauptspeicher (RAM).
- **Spyware:** Spyware versucht unauffällig im Hintergrund des Systems zu agieren und sensible Daten des Opfers an den Angreifer auszuliefern. Die Software versucht z.B. die Keyboard-Eingaben des Users mitzuschneiden und so an Passwörter

zu gelangen.

- **Adware:** Adware hat meist keine schädliche Wirkung für das Opfer. Sie versucht Werbung auf dem System einzublenden und darüber den Angreifer zu bereichern.
- **Ransomware:** Diese Art von Malware verschlüsselt Teile des Systems, um vom Opfer Geldbeträge für die Freigabe der Daten zu erpressen.
- **Bot:** Ein Bot ist ein Programm, das einen Remote Zugriff auf dem infizierten System einrichtet und es somit in das Botnetz eingliedert. Der Angreifer kann Befehle an die Hosts im Botnetz senden und sie so für z.B. DDoS-Angriffe oder das Versenden von Spam missbrauchen.
- **Hardware Damaging Malware:** Diese Art von Malware ist dadurch charakterisiert, die Hardware eines Systems zu beschädigen, z.B. das BIOS zu löschen oder Festplatten zu beschädigen.
- **Dropper:** Ein Dropper ist ein Programm, das selbst über keine Schadfunktion verfügt. Es dient lediglich zum Nachladen von weiterer Malware.
- **Exploit:** Exploits nutzen gezielt Schwachstellen in Programmen und Systemen aus, um sich Zugang zu ihnen zu verschaffen und eigenen Schadcode zu platzieren. Dieser wird, durch z.B. Umleitung des Programmablaufs, zur Ausführung gebracht. Meist wird über diese Methode Malware nachgeladen und so in Systeme eingeschleust [GDA17].

Linux ist im Bereich der Host-Betriebssysteme weitaus weniger verbreitet als Windows. Beliebte Angriffsziele für Malware sind daher Server, die mit Linux betrieben werden. Sie werden oft für die Ausbreitung von Botnetzen genutzt [KSK17]. Es folgen zwei Beispiele für Malware, die sich innerhalb von Linux-basierten IT-Systemen ausbreitet.

Das Bot-Netz „Mirai“

Ein Beispiel für so ein Botnetz ist die Schadsoftware „Mirai“, die ausschließlich Linux Systeme befällt. Es handelt sich dabei um einen Wurm, der das befallene System in das Botnetz integriert. Betroffen sind hiervon auch IoT-Geräte („Internet of Things“), die ebenfalls häufig auf einem Linux-Kernel laufen. Oft werden die vom Hersteller vergebenen Standardpasswörter nicht geändert, so dass sich der Wurm über Brute-Force leicht Zugang verschaffen kann [KB16].

Die Ransomeware „killDisk“

Die bösartige Software killDisk tauchte zum ersten Mal Ende 2015 in der Ukraine im Zusammenhang mit Angriffen auf den Energiesektor auf. Die Malware spezialisierte sich zunächst auf das endgültige Löschen von ganzen Festplatten, verfügt neuerdings aber über Optionen zur Verschlüsselung. Damit verhält sie sich wie Ransomware. Sie beschränkt sich nicht mehr nur auf Windows Systeme, sondern greift auch Linux-Workstations

und Server an. Die Besonderheit bei der Linux Variante ist, dass sie die Einträge des Bootloaders überschreibt und dort den Erpressertext abbildet [Arg17].

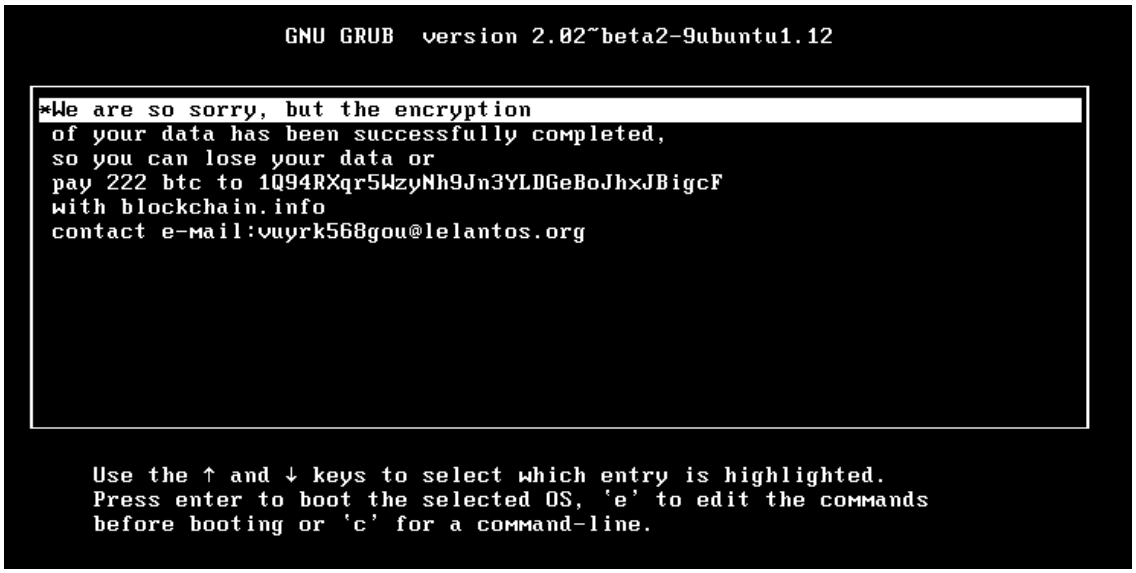

Abbildung 2.3: Die Erpressernachricht wird bei KillDisk im Bootloader angezeigt. [LK17]

2.3 Malware Analyse

Unter dem Begriff Malware Analyse werden Methoden zur Erkennung von Malware und Möglichkeiten zur semantischen Analyse von Schadcode zusammengefasst.

Ziel einer Malware Analyse ist es, eine Infizierung des IT-Systems mit Malware nachzuvollziehen. Ziel der Analyse ist es, ein Verständnis für folgende Themen aufzubauen [SH12, S. 34 ff.]:

- Welche Funktion erfüllt der Schadcode?
- Wie kann die Malware im System/Netzwerk erkannt werden?
- Wie viele Systeme wurden bereits infiziert?
- Wie kann die Schadwirkung eingedämmt werden?

Zur Beantwortung dieser Fragen gibt es verschiedene Strategien. Bei der Funktionsanalyse des Schadcodes unterscheidet man in zwei Ansätze: den *statischen* und *dynamischen* Ansatz.

Bei der *statischen Analyse* wird der vorliegende Malwarecode untersucht, ohne dabei zur Ausführung zu kommen. Der Analyst versucht den Code des Malware Entwicklers nachzuvollziehen und so die Funktion der Malware zu erfassen. Diese Methode ist jedoch sehr anfällig gegenüber Verschleierungstechniken, wie Obfuskation des Codes, Verschlüsselung oder Kompression von Malware Dateien [MKK07] [Hum17, S. 206 ff.].

Bei der *dynamischen Analyse* wird die Malware nicht mehr auf Ebene des Quellcodes untersucht, sondern während der Ausführung überwacht. Die Malware wird für die Analyse in einem abgesicherten Testsystem ausgeführt. So können unter anderem Verbreitungsstrategien und die konkrete Schadwirkung im System untersucht werden.

Aus der Analyse der Funktion des Codes können ebenfalls Strategien zur Abwehr bzw. Eindämmung der Schadwirkung abgeleitet werden. Wurde der gefundene Schadcode untersucht, muss anschließend das betroffene System bzw. Netzwerk auf weitere Verbreitungen der Malware untersucht werden. Bei der Erkennung von Malware unterscheidet man in folgende Strategien [Sch10]:

- Signatur-basierter Ansatz
- Verhaltens-basierter Ansatz
- Heuristischer Ansatz

Unter *Signatur-basierter* Erkennung von Malware versteht man das Finden von Malwareartefakten, anhand von typischen Bytefolgen oder Zeichenketten. Diese Signaturen können das Ergebnis einer statischen Analyse des Schadcodes sein. Die Methode ist abhängig davon, die Signatur einer Malware zu kennen und eignet sich nicht zum Finden von unbekannter Malware.

Verhaltens-basierte Erkennung von Malware (oder auch Behavioural Blocking) basiert auf dem Blockieren von unerwünschtem Verhalten. Ein Programm wird dazu permanent im Hintergrund des Systems ausgeführt und überwacht alle Prozesse. Wenn eine Aktivität einen Schwellwert übersteigt, wird sie als Malware erkannt und blockiert.

Bei einem *heuristischen* Ansatz werden Regeln über das Verhalten und die Funktion von Malware aufgestellt, mit denen ein System untersucht wird. Diese Regeln basieren auf den „Indicators of Compromise“ (siehe Abschnitt 2.4).

Es lässt sich auf Seiten der Funktionsanalyse von Malware eine weitere Unterscheidung in *einfache* und *fortgeschrittene* Analyse vornehmen [SH12, S. 34 ff.].

Bei der *einfachen statischen Analyse* wird Schadcode untersucht, ohne dabei auf die genauen Anweisungen einzugehen. Sie dient dazu, eine erste Aussage darüber zu treffen, ob eine vorliegende Datei infiziert oder harmlos ist. Genauere Informationen über das Verhalten können meist nicht erhoben werden. Zudem ist dieses schnelle Verfahren ineffektiv gegenüber höher entwickelter Malware, welche Verschleierungstechniken einsetzt.

Die *einfache dynamische Analyse* führt Schadcode in einer sicheren Testumgebung aus, um das Verhalten der Malware zu erforschen. Ziel ist es, einfache Signaturen abzuleiten und herauszufinden, wie die Malware von einem infizierten System entfernt werden kann.

Die *fortgeschrittene statische Analyse* versucht durch Reverse Engineering den Schadcode zu erfassen und die einzelnen Instruktionen zu verstehen. Hierfür kann der Code in einen Disassembler geladen werden, um ihn von Maschinencode in für Menschen lesbare Sprache zu übersetzen.

Im Gegensatz dazu die *fortgeschrittene dynamische Analyse*. Hier wird die Malware in einem Debugger untersucht, um Schritt für Schritt den inneren Zustand der infizierten Executable während der Ausführung zu dokumentieren.

Für eine effektive Malware Analyse ist eine Kombination der verschiedenen Ansätze nötig.

2.4 Indicators of Compromise

Als „Indicators of Compromise“ (IoC) werden Spuren in IT-Systemen bezeichnet, die mit einer hohen Wahrscheinlichkeit auf Malware hindeuten. Dies können zum Beispiel Viren-Signaturen in Form von Hashwerten oder bestimmte Dateinamen sein, die mit Malware in Verbindung gebracht werden.

Über die Indikatoren können erste Hinweise für die Existenz von Malware gefunden werden. IoCs beweisen jedoch nicht das Vorhandensein von Malware. Je mehr Indikatoren erfüllt sind, desto wahrscheinlicher ist es, dass sich tatsächlich Malware im System eingenistet hat.

IoCs können als boole’sche Ausdrücke formuliert werden, welche Charakteristiken von Malware beschreiben. Diese Ausdrücke sollten so allgemein formuliert sein, dass sie auch unterschiedlich modifizierte Versionen der gleichen Malware finden, aber speziell genug, um die Anzahl der falsch-positiven Treffer gering zu halten [Lee13].

Der Begriff IoC umfasst viele verschiedene Arten von Indicators, die eine unterschiedliche Komplexität aufweisen. Um diese dennoch einheitlich dokumentieren zu können, gibt es verschiedene Ansätze. Das Unternehmen Mandiant hat hierfür das OpenIoC Format, ein XML Template für die Dokumentation von IoCs in maschinenlesbarer Form, entwickelt. Die IoCs können aus dem Template in weitere Regeln für ein Intrusion Detection System (IDS), zum Beispiel in Yara-Rules, übersetzt werden.

Zur Identifizierung von Angriffen auf ein Computersystem können bestimmte Verhaltensweisen innerhalb des Systems als erste Ansätze und IoCs zur Erkennung des Angriffs dienen [Chi13]:

- **Ungewöhnlicher Datenverkehr:** Malware kommuniziert oft nach Außen, um Daten vom befallenen System weiterzuleiten oder weitere Instruktionen von einem

sogenannten Command and Control Server (C&C Server) zu erhalten. Durch Überprüfung der Netzwerkverbindungen auf bekannte C&C Server oder unzulässige Verbindungen können Kommunikationskanäle von Schadsoftware erkannt werden.

- **Unübliches Verhalten von privilegierten Accounts:** Malware versucht oft Root-Rechte auf dem System zu erlangen, um sich weiter ausbreiten zu können. Dies kann verhindert werden, indem Aktivitäten von Usern mit Root-Rechten überwacht werden. Indikatoren für eine schädliche Aktivität können sein: Login zu ungewöhnlichen Zeiten (außerhalb der Arbeitszeit), unrechtmäßige Zugriffe auf das System oder auf Dateien.
- **Geografische Unstimmigkeiten:** Wenn sich ein Account mit vielen verschiedenen IP-Adressen innerhalb von kurzer Zeit einloggt, kann dies ein Hinweis auf Malware sein. Auch Verbindungen von Orten aus, die in keinem Zusammenhang mit dem Unternehmen stehen, sind verdächtig.
- **Login-Auffälligkeiten:** Viele gescheiterte Logins sind Hinweis auf automatisierte Skripte, die mit Brute-Force Zutritt zum System erlangen möchten. Die Login-Versuche können dabei auch für nicht existente Accounts stattfinden. Wenn viele gescheiterte Zugriffe für einen bestimmten Account existieren, muss der Account auf Schadsoftware überprüft werden.
- **Gehäufte Datenbankzugriffe:** Wenn Schadsoftware versucht, Daten vom infizierten Rechner an den Angreifer auszuliefern, äußert sich das in erhöhten Lesezugriffen auf Datenbankentabellen.
- **Große HTML-Response:** SQL-Injection Angriffe auf eine Datenbank können sich durch vielfach größere HTML-Responses äußern. Sie werden genutzt, um innerhalb des Protokolls die Daten an den Angreifer weiterzuleiten.
- **Viele Anfragen für eine Datei:** Um an sensible Daten einer bestimmten Datei zu kommen, verwenden Angreifer verschiedene Angriffstechniken. Ein solcher Angriff ist durch das hohe Aufkommen von vielen Anfragen auf eine Datei, ausgehend von einer Quelle, zu erkennen.
- **Ungewöhnliche Ports:** Gerade Kommunikation von Botnetzten mit dem C&C Server findet häufig über Ports statt, die nicht Standard für die von Malware verwendeten Protokolle sind.

Diese Auffälligkeiten können in Regeln formuliert werden und gelten dann als IoC. Man unterscheidet in *Host-basierte Indikatoren* und *Netzwerk-basierte Indikatoren*. Host-basierte Indikatoren werden aufgestellt, um Malware auf einem einzelnen infiziertem System zu lokalisieren. Hierfür werden zum Beispiel der RAM oder einzelne Dateien auf dem Host untersucht. Netzwerk-basierte Indikatoren werden im Zusammenhang mit der Überwachung von Netzwerktraffic aufgestellt, um die Kommunikation von Malware aufzudecken [SH12].

Der Blogger David Bianco unterteilt IoCs in seinem Artikel „Pyramid of Pain“ [Bia14] in folgende Typen:

- **Hash-Werte:** Indikatoren auf der Basis von Hash-Werten sind leicht zu umgehen, da bereits kleinste Veränderungen in der Byteabfolge eine große Veränderung im Hash-Wert zur Folge haben.
- **IP-Adressen:** Indikatoren auf Basis von IP-Adressen können ebenfalls sehr leicht umgangen werden, da IP-Adressen leicht verändert werden können.
- **Domain Names:** Das Ändern von Domain Names ist etwas aufwendiger, da diese gekauft und registriert werden müssen. Dadurch, dass einige DNS Anbieter zu geringe Anforderungen an die Registrierung stellen, ist dieser Schritt immer noch sehr einfach.
- **Netzwerk und Host Artefakte:** Als Netzwerk und Host Artefakte wird von normalen Usern abweichendes Verhalten im Netzwerk oder auf einem Host bezeichnet. Die Indikatoren müssen zunächst von den Angreifern erkannt werden und anschließend die verwendeten Tools und Skripte angepasst werden. Die Änderungen im Code, um den Indikator zu umgehen, können gering sein, dennoch benötigt dieser Schritt sehr viel Zeit.
- **Tools:** Indikatoren, die von Angreifern verwendete Tools blockieren, sind sehr schwer zu umgehen. Die Angreifer müssen andere Tools finden, die die gleiche Funktion übernehmen oder eigene Software entwickeln.
- **Tactics, Techniques and Procedures (TTPs):** TTPs beschreiben komplexe Handlungsabläufe, die mit Malware in Verbindung stehen. Die Indikatoren beschränken sich dabei nicht nur auf ein verwendetes Tool, sondern beschreiben eine bestimmte Angriffsmethode. Indikatoren dieser Stufe sind aufgrund ihrer Komplexität für Angreifer nur sehr schwer zu umgehen.

Die Typen ordnet er in eine Pyramide ein, je nachdem, wie schwer die Indikatoren für Malware Entwickler zu umgehen sind (siehe Abb.:2.4).

2.5 Malwareverhalten in einer Linux-Umgebung

Angreifer entwickeln verschiedenste Arten von Malware, um ihre eigenen Interessen durchzusetzen. Für jeden Angriffsversuch bleibt dabei gleich, das Malware sich Zugriff auf das System verschaffen muss. Oft ist es notwendig, nach erfolgreicher Infizierung Root-Rechte zu erlangen, um kriminelle Interessen durchzusetzen. Des weiteren soll Malware einen Reboot überstehen und auch danach wieder auf dem System aktiv werden. Diese Grundsätze gelten nicht nur für Malware auf Linux, sondern allgemein für sämtliche Malware.

Viele Linux Distributionen profitieren davon, dass der Superuser „root“ standardmäßig gesperrt ist und privilegierte Useraccounts nur über das Kommando „sudo“ Programme mit Root-Rechten ausführen können [ubu15]. Eine hintergründige Installation von Malware über z.B. das bloße Anklicken von Email-Anhängen, wie unter Windows, ist damit nicht einfach möglich. Wie erlangt Malware unter Linux also Root-Privilegien?

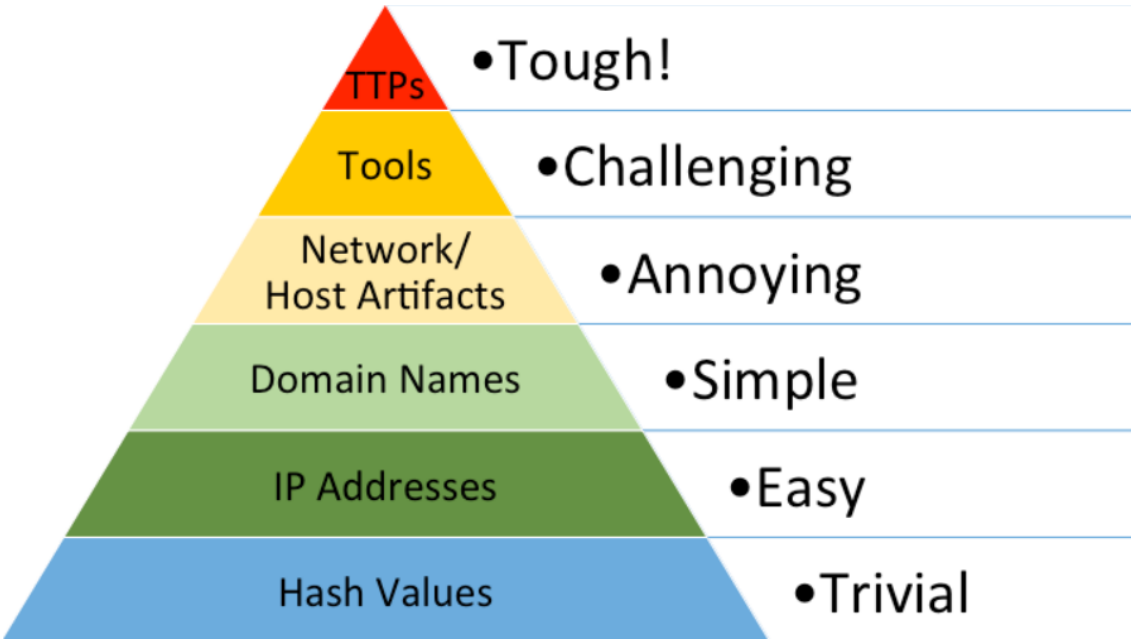

Abbildung 2.4: IoCs geordnet nach Aufwand, der von Malware Entwicklern betrieben werden muss, um diese zu umgehen. [Bia14]

- **Social Engineering:** Der Malware Entwickler muss, z.B. den Administrator, dazu bringen Malware auf dem System zu installieren.
- **Exploits:** Netzwerkdienste, die mit Root-Rechten laufen, können über ein Exploit ausgenutzt werden und sich darüber Zugriff zum System verschaffen.
- **Rechteerweiterung** (eng. *priviledge eskalation attack*)**:** Einen Account übernehmen und mit Angriffen zur Rechteerweiterung Root-Rechte erlangen.

Anschließend kann ein Rootkit installiert werden, um zukünftig unter Umgehung der normalen Zugriffskontrolle Root-Rechte zu besitzen und die Aktivitäten des Angreifers auf dem System zu verbergen. Dazu zählt das verstecken von schädlichen Prozessen und Dateien, sowie das Manipulieren von Log-Dateien [LCLW14, S. 721].

Rootkits gelangen über zusätzlich geladene Kernelmodule in den Kernel. Diese können zu Laufzeit integriert werden und besitzen vollen Zugriff auf die Kernel API. Anschließend wird das Kernelmodul aus der Liste der geladenen Module im Kernel gelöscht, um es im System zu verstecken [LCLW14, S. 722].

Über das installierte Rootkit können Systemcalls manipuliert werden (siehe Abb. 2.5). Ruft ein Prozess den veränderten Systemcall auf, können falsche Parameter zurückgegeben werden oder falsche Informationen über das System verbreitet werden [LCLW14, S. 734].

Wesentlich einfacher ist die Infizierung von „Internet of Things“ Geräten (IoT), wie Routern, IP-Kameras oder Geräten aus der Smart-Home-Infrastruktur. Die Angriffe basieren

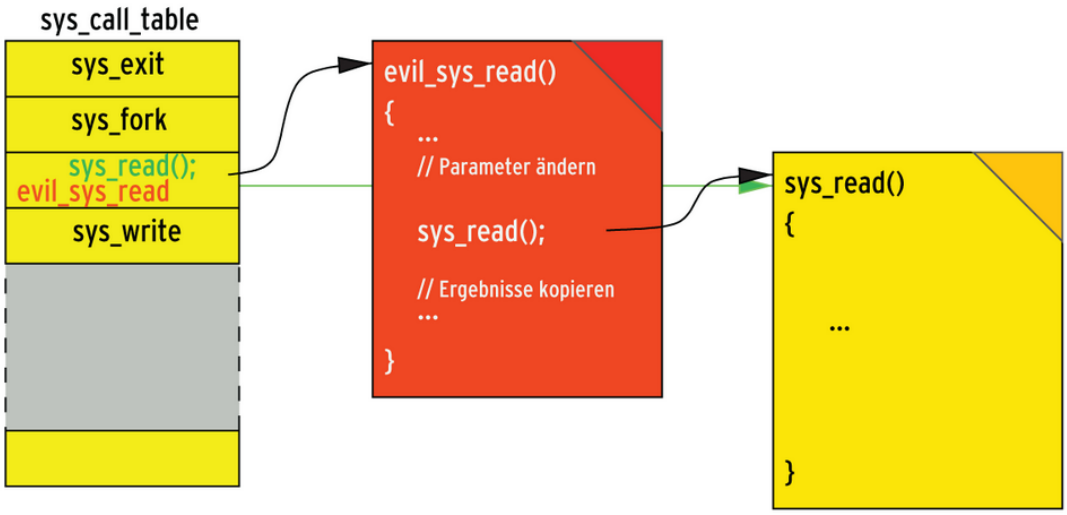

Abbildung 2.5: Manipulation eines Systemcalls [Qua12]

auf dem Ausprobieren von Standard-Kombinationen von Passwörtern und Usern, so wie sie von den Herstellern festgelegt wurden. Die Standard-Passwörter werden meist nicht mehr geändert, so dass sich große IoT-Botnetze organisieren können. Hier gibt es bereits eine Gegenbewegung in Form der Malware „Brickerbot“, die ebenfalls IoT-Geräte befällt, deren Speicher löscht und die Netzwerkverbindung trennt. Somit soll eine weitere Ausbreitung von den Botnetzen unterbunden werden [Tri17].

Um auch nach einem Reboot die Malware automatisch ausführen zu lassen, verändert die Malware Skripte, die während des Bootvorgangs aufgerufen werden, in Skripte, die den Schadcode ausführen. Eine weitere Möglichkeit, um Persistenz zu erlangen, ist die Veränderung der Autostartregion der einzelnen User. In diesen wird festgelegt, welche Programme beim Login eines Users ausgeführt werden. So können Angreifer beispielsweise Ihre Skripte gezielt beim Login des Administrators ausführen lassen [SS17].

2.6 IoC Analyse unter Windows

Die Automatisierung der Schritte zur Suche nach Malwarespuren unter Windows konnte mit mehreren Skripten umgesetzt werden. Es konnten die Schritte der Datenerhebung und der Bewertung, ob ein IoC auf das System zutrifft, automatisiert werden. Die endgültige Entscheidung, ob und welche Malware vorliegt, kann hingegen nur von Malware Analysten getroffen werden.

Das Skript für die Untersuchung von Windows Systemen setzt sich aus zwei Teilen zusammen. Im ersten Teil wird ein Skript zur Datenerhebung direkt auf dem betroffenen

Rechner ausgeführt. Das Skript erhebt ein RAM-Image und sammelt weitere Informationen, wie z.B. welche Programme in den Autostarts von Windows vermerkt sind. Gleichzeitig wird ein Scan nach Malware auf dem System durchgeführt.

Das erstellte RAM-Image wird anschließend von einem weiteren Skript auf Malware Artefakte untersucht (siehe Abb.: 2.6). Hierfür startet das Skript verschiedene Module des Frameworks Volatility , welche das RAM-Image auswerten. Die Ergebnisse von Volatility werden in einer SQLite Datenbank gespeichert. Durch gezielte SQL-Abfragen können die wichtigsten Informationen aus den Ergebnissen extrahiert und für die manuelle Auswertung nachbereitet werden. Dies umfasst z.B. die Ausgabe von allen aktiven Netzwerkverbindungen oder Prozessen zum Zeitpunkt der Erstellung des Images.

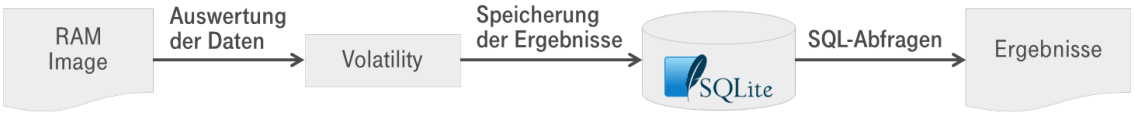

Abbildung 2.6: Schematischer Aufbau des Skriptes (Quelle: eigene Darstellung)

3 Methode

Im folgenden Kapitel sollen die Ansätze zur Erstellung einer detaillierten IoC-Übersicht und Einschätzung der IoCs bezüglich Automatisierbarkeit präsentiert werden.

Um im Ergebnis eine Übersicht mit IoCs für Linux-Malware erstellen zu können, muss zunächst eine vorbereitende Recherche erfolgen. Durch die gelegten Wissensgrundlagen können im Anschluss eigene IoCs definiert werden. Innerhalb der Übersicht erfolgt eine Einschätzung, inwiefern sich die Überprüfung auf den jeweiligen Indikator automatisierten lässt. Eine Bestätigung der Schlussfolgerungen erfolgt durch die Umsetzung der Ansätze innerhalb eines Skriptes. Um zu testen, ob die aufgestellten IoCs Malware erkennen können, wird für ausgewählte Indikatoren ein Testszenario entworfen.

3.1 Recherche

Zu Begin der Recherche muss der Begriff „Indicator of Compromise“ genauer definiert werden. Es wirde nach Möglichkeiten recherchiert, die einzelnen IoCs zu dokumentieren. Hierfür werden technische Berichte von IT-Sicherheitsfirmen über spezifische Malware gesichtet. So zum Beispiel der Bericht über die Malware-Kampagne „Windigo“ der Firma ESET [ese14]. Innerhalb des Anhangs dieser Berichte findet sich eine Auflistung von verschiedenen IoCs, mit Hilfe derer die beschriebene Malware erkannt werden kann.

Zusätzlich wird Wissen zu typischer Malware unter Linux gesammelt. Hierfür werden Malware Datenbanken nach Linux Malware abgesucht. Die Ergebnisse finden sich im Abschnitt 4.2.1.

Die Berichte einzelner Linux-Malware werden gesichtet, um daraus auf das Verhalten der Malware innerhalb des Betriebssystems schließen zu können. Interessant sind hierbei die Strategien der Malware beim initialen Befall (Angriffsvektoren)des Sytems, Möglichkeiten des Erlangens von administrativen Rechten und Persistenzstrategien. Zuvor erfolgt eine Ausseinandersetzung mit dem Aufbau des Linux Betriebssystems, so dass Bereiche des Betriebssystems bestimmt werden können, die häufig von Malware verändert werden. Weitere Informationen, wie sich Malware innerhalb von Linux verhält, können aus Büchern zum Vorgehen bei einer Malware Analyse [SH12] [LAHR11] und Literatur zur forensischen Hauptspeicheranalyse [LCLW14] abgeleitet werden. Das Vorgehen bei einer Malware Analyse bezieht sich in den Büchern immer auf die Analyse eines Windows System, da es kaum Forschung oder Literatur gibt, die sich mit Malware unter Linux auseinandersetzt. Durch das zuvor angeeignete Wissen über den Aufbau des Linux Betriebssystems, können Parallelen zwischen Windows und Linux gezogen werden. Weitere Quellen für die Malware Erkennung unter Windows sind die SANS Poster „Finding Un-

known Malware“ [Lee13] und „Know Abnormal...Find Evil“ [PL16]. Die Poster geben Hinweise darauf, das eine automatische Malware Erkennung mit IoCs möglich ist und beschreiben einige IoC für Windows Hosts. Diese können teilweise auf Linux übertragen werden.

Um die Überprüfung eines Systems mit IoC automatisieren zu können, mussten Ansätze zur Umsetzung auf Linux recherchiert werden. Es wird nach Software recherchiert, die bereits Funktionen zur automatisierten Suche beinhaltet oder diese unterstützen kann. Schwerpunkt liegt hierbei auf der Suche nach einem guten Malware Scanner für Linux, Software zur Hauptspeicheranalyse und Tools, die die automatisierte Datenerhebung übernehmen.

3.2 Aufbau der IoC-Übersicht

Durch die Recherche wurde die nötige Wissensgrundlage geschaffen, um IoCs für Linux abzuleiten. Die IoCs sollten möglichst allgemein formuliert sein, so dass sie nicht nur eine bestimmte Art oder Familie von Malware beschreiben, sondern innerhalb kurzer Zeit eine Aussage getroffen werden kann, ob auf einem System Malware vorliegt. Hierfür ist eine genaue Bestimmung der Malware mit Hilfe der IoCs nicht nötig.

Die Entscheidung, welche Malware auf dem System vorliegt, kann von einem Malware Analysten anhand der gefunden Spuren getroffen werden. Um die aufgelisteten IoCs später in einem Skript umsetzen zu können, erfolgt innerhalb der Bachelorarbeit eine Vorüberlegung bezüglich der Automatisierbarkeit. Die Automatisierbarkeit bezieht sich hierbei zum einen auf die Erhebung der Datengrundlage, zum anderen auf die anschließende Auswertung der erhobenen Daten. Für beide Fälle muss eine Einschätzung hinsichtlich der Realisierbarkeit getroffen werden. Hierzu zählt eine Vorüberlegung, ob und wie die Automatisierung umgesetzt werden kann. Der ausgearbeitete Ansatz soll innerhalb der detaillierten Einzelbeschreibung des IoC ausformuliert werden.

Die unter diesen Einschränkungen erstellten IoCs, sollen im nächsten Schritt dokumentiert werden. Hierzu zählt eine Beschreibung des IoCs, die beinhaltet, warum es sich dabei um einen „Indicator of Compromise“ handelt. Hierfür wird aufgeführt, wie der IoC bei einer Infizierung durch Malware entsteht und welche Malware-Klasse dieses im besonderen verursacht. Für jeden Indikator erfolgt eine Entscheidung, ob die Erkennung automatisiert werden kann. Ist die Automatiserung möglich, soll in der Beschreibung ein Ansatz angegeben werden, wie diese erfolgen könnte.

3.2.1 Automatisierung der Datenerhebung

Für jeden gefundenen IoC muss definiert werden, welche Daten benötigt werden, um entscheiden zu können, ob die Kriterien des Indikators erfüllt sind. Für die Analyse wird

die Arbeit auf einem Abbild des Systems empfohlen. Hierbei stellt sich die Frage, ob ein RAM-Image genügt, ein Triage Image nötigt ist oder das Live-System bzw. ein Abbild des Live-Systems herangezogen werden muss. Um die Realisierbarkeit der Automatisierung dieses Schrittes zu bestätigen, kann Software aufgeführt werden, deren Funktion die geforderten Arbeitsschritte beinhaltet.

3.2.2 Automatisierung der Auswertung der erhobenen Daten

Bei der Entscheidung hinsichtlich der Auswertung der erhobenen Daten muss ebenfalls auf Automatisierbarkeit überprüft werden. Durch eine Analyse der erhobenen Daten ist zu klären, ob der Indikator auf die Daten zutrifft. Ist dies der Fall, ist dies kein Beweis für die Existenz von Malware, da immer eine Wahrscheinlichkeit für einen falsch-positiven Treffer besteht. Die endgültige Entscheidung darüber, ob Malware vorliegt, sollte von Fachpersonal anhand der gefundenen Spuren entschieden werden.

Sollte für einen Indikator die Entscheidung getroffen werden, dass sich dieser nicht automatisiert auswerten lässt, muss dies begründet werden.

3.2.3 Dokumentation der Indikatoren

Die unter diesen Kriterien aufgestellten IoCs werden zunächst in einer tabellarischen Übersicht (siehe Tab.:3.1) dokumentiert. Die Tabelle soll einen ersten Überblick über die erstellten IoCs geben und die Entscheidungen bezüglich der Automatisierbarkeit enthalten. Jeder IoC wird einer Kategorie zugeordnet, die beschreibt, auf was sich der Indikator bezieht. Die Kategorien ergeben sich aus den vorliegenden Daten und sind vom Autor selbst gewählt.

Zu jedem Eintrag in der Tabelle soll ein separater Textabschnitt existieren, in dem der IoC detailliert beschrieben wird. Dieser unterliegt folgender Grundstruktur:

Bezeichnung des IoC

Beschreibung: *Was macht den Indikator aus? Warum ist der Indikator ein Hinweis auf Malware?*
autom. Datenerhebung: *Ja\|Nein.*
autom. Auswertung: *Ja\|Nein.*
Ansatz Datenerhebung: *Beschreibung des Ansatzes zur Datenerhebung. Wenn die Datenerhebung nicht automatisiert möglich ist, muss zudem begründet werden, wo die Probleme bei der Umsetzung liegen.*
Ansatz Auswertung: *Beschreibung des Ansatzes zur Auswertung. Wenn die Auswertung nicht automatisiert möglich ist, muss zudem begründet werden, wo die Probleme bei der Umsetzung liegen.*

Die Einträge der Tabelle sind mit dem dazugehörigen Textabschnitt verlinkt.

Tabelle 3.1: Struktur der IoC-Übersicht

IoC	autom. Datenerhebung	autom. Auswertung	Kategorie
Bez. des IoC	(*ja|nein)*	(*ja|nein*)	*Bez. der Kategorie*

3.3 Test der aufgestellten Indikatoren

Um zu testen, ob die automatisierte Erkennung von Malware-Spuren in einer Linux-Umgebung möglich ist, soll ein Testszenario entworfen werden. Der Test soll bestätigen, dass die Erkennung von Malware anhand der gefundenen IoCs automatisiert werden kann. Um dies zu realisieren, soll der Automatisierungsansatz, für ausgewählte Indikatoren, in einem Skript umgesetzt werden. Dieses soll sowohl Funktionen zur automatischen Datenerhebung, als auch Funktionen zur automatischen Analyse der Daten besitzen. Die Ansätze für die Automatisierung werden aus der Beschreibung der Indikatoren bezogen.

Das so entstandene Skript soll dabei nur lesend auf das zu untersuchende System zu greifen, um Veränderungen zu vermeiden. Dabei soll die Ausführung möglichst unabhängig von der vorliegenden Linux Distribution sein. Die Ausführung der einzelnen Funktionen soll möglichst automatisiert ablaufen.

Das zu testende System wird durch eine virtuelle Maschine dargestellt. Sie soll mit verschiedener Malware kompromittiert werden. Anschließend wird mit Hilfe des Skriptes überprüft, ob bestimmte Indikatoren erfüllt sind und somit auf Malware hindeuten.

Als Quelle für Malware Samples sollen öffentliche Malware Datenbanken genutzt werden.

4 Ergebnis

In dem folgenden Kapitel werden die gefundenen Indikatoren vorgestellt. Zudem wird der Test beschrieben, mit dem die Umsetzung der automatisierten Lösung aufgezeigt werden soll.

4.1 IoC-Übersicht

Für die IoC-Übersicht wurde eine Tabelle angelegt. Zu jedem Eintrag der Tabelle existiert ein Textabschnitt, der das IoC näher beschreibt.

4.1.1 Tabellarische Übersicht

Insgesamt wurden vom Autor 26 IoCs aufgestellt und 6 verschiedenen Kategorien zugeteilt. Folgende Kategorien werden unterschieden:

- *1 - User*: bezieht sich auf Manipulationen von Usern oder deren Rechten
- *2 - Prozesse*: bezieht sich auf die Erkennung von schädlichen Prozessen
- *3 - Kernel*: bezieht sich auf Veränderungen innerhalb des Linux-Kernels, als Folge von Malware Aktivitäten
- *4 - Datei*: bezieht sich auf Manipulationen von Dateien, die durch Malware hervorgerufen werden
- *5 - Netzwerk*: bezieht sich auf Veränderungen im Netzwerkverkehr, die durch Malware ausgelöst wurden
- *6 - Pakete*: bezieht sich auf von Malware hervorgerufenen Veränderungen bei Paketen

Die Kategorien wurden erst nach der Definition von IoCs aufgestellt. Sie richten sich nach dem jeweiligen thematischen Schwerpunkt der Indikatoren. Innerhalb der Kategorien werden die IoCs ebenfalls durchnummeriert. Die ID setzt sich folgender Maßen zusammen: *Kategorie - Nummer des IoC*

Tabelle 4.1: IoC-Übersicht

ID	**IoC**	**Autom. Datenerhebung**	**Autom. Auswertung**
1-1	Nicht vorgesehene User	j	j
1-2	Unberechtigte User in Gruppen mit Admin-Rechten	j	j

Tabelle 4.1: IoC-Übersicht

ID	**IoC**	**Autom. Datenerhebung**	**Autom. Auswertung**
1-3	Systemuser mit Kennwort und/oder Login-Shell	j	j
1-4	User „nobody“ hat mehr Rechte als nobody	j	j
2-1	Abweichung von: init Prozess hat pid 1	j	j
2-2	Systemprozesse von Userprozessen gespawnt	j	j
2-3	Versteckte Prozesse	j	j
2-4	Schädliche init Skripte	j	n
2-5	Schädliche Cronjobs	j	n
2-6	Schädliche Autostartskripte	j	n
2-7	Abweichende Aufrufparameter	j	n
3-1	Manipulierte Systemcalls	n	j
3-2	Manipulierte Kernelmodule	j	n
4-1	Hashwert auf Blacklist	j	j
4-2	Malware Dateinamen	j	j
4-3	Executables an ungewöhnlichen Orten	j	n
4-4	Statisch gelinkte Dateien	j	j
4-5	Content Extension Mismatch	j	j
4-6	Linux untypische Extensions	j	j
4-7	UPX gepackte Executables	j	n
4-8	Applikationen legen Dateien an ungewöhnlichen Orten ab	n	n
5-1	Ungewöhnliche Ports/ Dienste	j	n
5-2	Malware IP-Adressen	j	j
5-3	Malware Domain Names	n	j
6-1	Nicht verifizierte Pakete	j	j
6-2	Nicht vorgesehene Pakete	n	n

4.1.2 Kategorie 1: User

Indikatoren dieser Kategorie beziehen sich auf Malware-Spuren, die mit Veränderungen bei Usern oder deren Rechten zu tun haben.

1-1 Nicht vorgesehene User

Beschreibung: Die Kriterien des Indikators sind erfüllt, wenn sich auf dem zu untersuchendem System für dieses nicht vorgesehene User befinden. Malware versucht über zusätzlich erstellte User, Backdoors für einen erneuten Zugriff mit Umgehung der Zugangskontrolle zu erstellen. Für die Erstellung von Usern werden administrative Rechte benötigt. Verfügt Malware bzw. der Angreifer über administrative Rechte, kann er sich uneingeschränkt über das ganze System ausbreiten.
autom. Datenerhebung: Ja.
autom. Auswertung: Ja.
Ansatz Datenerhebung: Aus der Datei */etc/passwd* können alle User des Systems extrahiert werden. Problematisch wird es, wenn die User nicht auf dem System registriert sind, sondern sich über Authentifizierungsdienste, wie Kerberos anmelden. Diese sind nicht in */etc/passwd* gelistet. In Absprache mit den Systemverantwortlichen, ist eine Liste mit allen für das System vorgesehenen Usern zu erstellen. Wenn Backupfiles vorhanden sind, können sie ebenfalls zum Abgleich herangezogen werden. Dies ist nur sinnvoll, wenn ausgeschlossen werden kann, dass die Backupfiles ebenfalls infiziert wurden.
Ansatz Auswertung: Die aus dem Gespräch mit den Systemverantwortlichen erstellte Whitelist, wird mit der aus dem aktuellen System erstellten Liste abgeglichen. Unter Linux kann dies mit dem Kommando *diff* umgesetzt werden. Wenn Unterschiede erkannt werden, gilt der Indikator als erfüllt.

1-2 Unberechtigte User in Gruppen mit Admin-Rechten

Beschreibung: User, die über *sudo* Rechte verfügen, dh. Programme mit den Rechten anderer User ausführen können oder User, die unberechtigt über administrative Rechte verfügen, sind ein Zeichen für eine Manipulation des Systems durch einen Angreifer. Über die Erweiterung der Rechte eines Users, der sich unter Kontrolle der Malware (bzw. des Angreifers) befindet, kann sich Malware weiter über das System ausbreiten oder ein Backdoor für den Angreifer schaffen, um zu einem späteren Zeitpunkt erneut über administrative Rechte auf dem System zu verfügen.
autom. Datenerhebung: Ja.
autom. Auswertung: Ja.
Ansatz Datenerhebung: In der Datei */etc/group* sind alle Gruppen mit den dazugehörigen Usern gelistet. Zusammen mit den Systemverantwortlichen kann eine Liste der sudo-berechtigten User erstellt werden. Die User aller Gruppen mit administrativen Privilegien, wie sudo/admin und root, können aus */etc/group* extrahiert werden.

Ansatz Auswertung: Die Whitelist kann mit den Daten aus dem aktuellen System abgeglichen werden. Der Vergleich von zwei Dateien auf Unterschiede kann unter Linux mit dem Befehl *diff* realisiert werden. Zudem muss sichergestellt werden, dass der Superuser root das einzige Mitglied der Gruppe root ist.

1-3 Systemuser mit Kennwort und/oder Login-Shell

Beschreibung: Systemuser werden von bestimmten Diensten benötigt, um Dateien zu lesen oder auszuführen. Diese besitzen keine Kennwörter, da sie sich nicht auf dem System einloggen, wie ein normaler User. Demzufolge benötigt ein Systemuser auch keine Login-Shell. Systemuser können von Malware übernommen werden, um Zugriff auf bestimmte Dienste im System zu bekommen. Zudem ist eine Übernahme eines bestehenden Users unauffälliger als das erstellen eines neuen Accounts.
autom. Datenerhebung: Ja.
autom. Auswertung: Ja.
Ansatz Datenerhebung: Für die Überprüfung der Systemuser muss zunächst eine Liste mit den gängigsten Systemusern für Linux erstellt werden. In der Datei */etc/passwd* können dann auf Grundlage dieser Liste alle Systemuser aufgelistet werden. In der */etc/shadow* sind die Passwörter der User in Form von Hashes aufgelistet.
Ansatz Auswertung: Für alle Systemuser ist als Login-Shell */sbin/nologin* oder */bin/false* eingetragen, da sie keine Login-Shell benötigen. Über */etc/shells* können alle gültigen Login-Shells des Systems eingesehen werden. Diese sollten für keinen Systemuser gesetzt sein. In der */etc/shadow* sollte für das Passwort ein „*“ oder „!“ gesetzt sein.

1-4 User nobody hat mehr Rechte als nobody

Beschreibung: Der User nobody ist ein Systembenutzer, der ein Minimum an Rechten besitzt. Er wird der Gruppe „other“ oder „nogroup“ zugeordnet, da diese die geringst mögliche Anzahl an Rechten besitzt. Der User besitzt keine Dateien oder Verzeichnisse. Abweichungen dieses Standards können auf eine Übernahme des Users durch Malware hindeuten.
autom. Datenerhebung: Ja.
autom. Auswertung: Ja.
Ansatz Datenerhebung: Im kompletten System muss geprüft werden, ob nobody Besitzer von Dateien oder Verzeichnissen ist. Dies kann mit *find / -user nobody* gemacht werden. Das Kommando sucht im gesamten Verzeichnisbaum nach Dateien, bei denen der User „nobody“ als Besitzer angegeben ist.
Ansatz Auswertung: Als Ergebnis der Suche nach Dateien von dem User nobody sollten bei einem sauberen System keine Treffer gefunden werden. Bei älteren Linux Systemen kann es zu Ausnahmen kommen, da früher Deamons unter dem User nobody liefen.

4.1.3 Kategorie 2: Prozess

Indikatoren dieser Kategorie behandeln Veränderungen in der Prozesshierarchie oder Abweichung des Verhaltens bestimmter Standardprozesse.

2-1 Abweichung von: init Prozess hat pid 1

Beschreibung: Der init Prozess wird am Ende des Bootvorgangs gestartet und hat die pid = 1. Er ist die Wurzel des Prozessbaum, d.h. alle Prozesse lassen sich auf ihn zurück führen. Alle Kindprozesse ohne Elternprozess werden von init adoptiert. Malware versucht schädliche Prozesse wie legitime Systemprozesse zu benennen, um sich im System zu verstecken. Wenn ein weiterer init Prozess existiert, wurde dieser von Malware erzeugt.
In den meisten Distributionen wurden mittlerweile Alternativen zu init eingeführt, da sie das Wechseln von Hardware im laufenden Betrieb unterstützen. Die am weitesten verbreitetste Alternative ist „systemd“, die unter Debian, Ubuntu und Fedora zum Einsatz kommt.
autom. Datenerhebung: Ja.
autom. Auswertung: Ja.
Ansatz Datenerhebung: Unter Linux lassen sich alle Prozesse mit dem Befehl *ps -ef* anzeigen. Mit dem Tool *grep* kann nach dem init-Prozess gesucht werden. Mit dem Tool *pstree* kann die Prozesshierarchie dargestellt und Beziehungen zwischen Prozessen nachvollzogen werden.
Ansatz Auswertung: Der init-Prozess sollte nur einmal auf dem System vorkommen. Bei mehrfachem Auftritt muss die PID des Prozesses überprüft werden. Diese sollte 1 sein. Bei einem anderen Wert handelt es sich um einen schädlichen Prozess, der als init getarnt ist.

2-2 Systemprozesse von Userprozessen gespawnt

Beschreibung: Systemprozesse sind Kindprozesse des init Prozesses, dh. die PPID (Parent Process ID) ist 1 (init hat pid = 1). Als Besitzer sidn immer Systemnutzer angegeben. Malware benennt eigene schädliche Prozesse wie Systemprozesse, um sie legitim wirken zu lassen und somit unerkannt im System zu bleiben.
autom. Datenerhebung: Ja.
autom. Auswertung: Ja.
Ansatz Datenerhebung: Unter Linux lassen sich alle Prozesse mit dem Befehl *ps -ef* anzeigen. Mit dem Tool *grep* kann nach den einzelnen Prozessen gesucht werden. Mit dem Tool *pstree* kann die Prozesshierarchie dargestellt und Beziehungen zwischen Prozessen nachvollzogen werden.
Ansatz Auswertung: Die erhobene Übersicht der Prozesse muss auf mehrfach vorkommende Prozesse überprüft werden. Mehrfach vorkommende Prozesse, deren PPID nicht

auf den init Prozess verweist, sind ein Indikator für von Malware erstellter Prozesse.

2-3 Versteckte Prozesse

Beschreibung: Malware versucht, um ihre Aktivitäten zu verbergen, Prozesse zu verstecken, so dass diese bei der Auflistung aller aktiven Prozesse nicht mehr auftauchen.
autom. Datenerhebung: Ja.
autom. Auswertung: Ja.
Ansatz Datenerhebung: Um von einem System die aktiven Prozesse untersuchen zu können, muss ein RAM-Image erzeugt werden. Anschließend müssen die aktiven Prozesse aus dem Hauptspeicherabbild ausgelesenen werden. Dies kann mit den Modulen *linux_pidhashtable* und *linux_pslist* des Volatility Framwork erfolgen. Als Ergebnis geben die Module jeweils eine Liste der gefundenen Prozesse aus.
Ansatz Auswertung: Da die verwendeten Module von Volatility unterschiedliche Ansätze zur Erstellung der Prozess-Übersicht verwenden, kann es vorkommen, das ein Prozess nur in einer der Übersichten auftaucht. Daher müssen die Übersichten geordnet und miteinander verglichen werden. Dies kann mit dem Tool *diff* unter Linux realisiert werden. Finden sich Unterschiede ist dies ein Hinweis auf einen versteckten Prozess.

2-4 Schädliche init Skripte

Beschreibung: Nach dem Booten des Systems ruft der init-Prozess in einer festgelegten Reihenfolge verschiedene Skripte auf. Diese Skripte werden oft von Malware manipuliert, um sich selbst nach dem Booten neu zu starten. Diese Taktik wird von Malware genutzt, um Persistenz im System zu erlangen. Ohne diesen Mechanismus würde die Malware, nach einem Reboot, nicht mehr aktiv sein und müsste von außen wieder neu initiiert werden.
autom. Datenerhebung: Ja.
autom. Auswertung: Nein.
Ansatz Datenerhebung: Bei den vom init-Prozess gestarteten Skripten handelt es sich um Shell-Skripte. Sie liegen unter */etc/init.d*.
Ansatz Auswertung: Die Dateien können mit einem Malwarescanner oder mit einer Keywordsuche nach Skripten durchsucht werden, die schädliche Executables aufrufen. Der Vorgang ist jedoch zu komplex, um automatisiert gute Ergebnisse zu liefern. Eine Betrachtung durch Fachpersonal ist für eine gute Einschätzung nötig.

2-5 Schädliche Cronjobs

Beschreibung: Cronjobs sind Skripte, die regelmäßig zu festgelegten Zeiten ausgeführt werden. Malware manipuliert Cronjobs, um Persistenz im System zu erlangen, indem der Cronjob zu einer bestimmten Zeit die Malware Executable ausführt. Unter dem Cron-Deamon laufen auch die sogenannten atjobs. Diese werden im Unterschied zu Cronjobs nur einmalig ausgeführt und können ebenfalls manipuliert werden.
autom. Datenerhebung: Ja.
autom. Auswertung: Nein.
Ansatz Datenerhebung: Die cron- und at-Dienste liegen für jeden User unter */var/spool/cron* und für System-weite Cronjobs unter */etc/cron.d*.
Ansatz Auswertung: Diese Orte können mit einem Malwarescanner überprüft werden, um eine erste Einschätzung treffen zu können. Um eine endgültige Entscheidung zu treffen, ob schädliche Skripte vorliegen, muss eine manuelle Analyse vorgenommen werden.

2-6 Schädliche Autostartskripte

Beschreibung: Autostartskripte können allgemein beim Login der User oder beim Login eines speziellen Users aufgerufen werden. Malware kann dies ausnutzen, um sich zum Beispiel beim Login des Administrators informieren zu lassen.
autom. Datenerhebung: Ja.
autom. Auswertung: Nein.
Ansatz Datenerhebung: Die Skripte befinden sich unter */etc/profile.d*, zudem werden die Dateien */etc/profile* und */etc/bash.bashrc* bei jedem Login aufgerufen. Zusätzlich existieren für jeden Account Konfigurationsdateien:

- *~/.bashrc*: Wird bei jedem Start einer interaktiven Shell (Terminal) ausgeführt.
- *~/.bash_profile*[3] Wird bei jedem Start einer Login Shell ausgeführt.
- *~/.config/autostart*: Programme, die nach der Anmeldung des Users, bei Verwendung einer grafischen Oberfläche, ausgeführt werden.

Ansatz Auswertung: Diese können mit einem Malwarescanner auf schädliche Inhalte gescannt werden, sollten aber manuell von Fachpersonal überprüft werden.

2-7 Abweichende Aufrufparameter

Beschreibung: Prozesse werden mit bestimmten Parametern aufgerufen. Werden diese verändert, kann dies ein Hinweis auf eine Manipulation durch Malware sein. Diese kann legitime Prozesse verändern oder selbst schädliche Prozesse erstellen, die nur anhand von abweichenden Aufrufparametern erkannt werden können.

[3] Alternative Schreibweisen: *.profile* oder *.bash_login*

autom. Datenerhebung: Ja.
autom. Auswertung: Nein.
Ansatz Datenerhebung: Die Kommandozeile, mit der ein Prozess aufgerufen wurde, wird für jeden Prozess unter */proc/cmdline* gespeichert. Wenn von dem betroffenen System ein RAM-Image erhoben wurde, kann dieses mit Hilfe von Volatility Modulen ausgewertet werden. Das Modul *linux_psaux* zeigt alle aktiven Prozesse mit Kommandozeile an, so dass die Aufrufparameter extrahiert werden können.
Ansatz Auswertung: Ein Ansatz, um die Aufrufparameter zu überprüfen, ist es Wissen über typische Aufrufe von bestimmten Prozessen zu sammeln und dieses mit der erhaltenen Liste abzugleichen.

4.1.4 Kategorie 3: Kernel

Indikatoren dieser Kategorie beschreiben Abweichungen von Abläufen oder Code innerhalb des Kernels.

3-1 Manipulierte System Calls

Beschreibung: Systemcalls werden von Prozessen genutzt, um bestimmte Funktionen, wie das Lesen einer Datei, vom Kernel ausführen zu lassen. Sie können von Malware manipuliert werden, um falsche Informationen über das System weiterzugeben. Hierfür wird in der Systemcall Tabelle der Pointer, der auf den Code im Kernel zeigt, geändert und zeigt z.B. auf Code in einem LKM (loadable Kernel Module). Eine weitere Möglichkeit die System Calls zu manipulieren, ist den Code im Kernel zu überschreiben. Der Pointer zeigt weiterhin auf den Kernel.
autom. Datenerhebung: Nein.
autom. Auswertung: Ja.
Ansatz Datenerhebung: Der Speicherort der Systemcall Tabelle ändert sich von Version zu Version des Kernels und muss zuerst ermittelt werden.
Mögliche Orte sind:

- */usr/include/syscall.h*
- */usr/include/x86_64-linux-gnu/asm/unistd_32.h* für 32-bit Plattformen
- */usr/include/x86_64-linux-gnu/asm/unistd_64.h* für 64-bit Plattformen

Ansatz Auswertung: Im Volatility Framework existiert das Modul „linux_check_syscall“, um die Systemcalls auf Veränderungen der Pointer zu überprüfen. Ihm wird zuerst der Pfad zu der Systemcall Tabelle übergeben. Die Ausgabe muss nach dem Keyword „HOOKED“ gefiltert werden; dies geht unter Linux am besten mit dem Befehl *grep*. Das Volatility Modul „linux_check_inline_kernel“ überprüft den Code im Kernel auf Manipulationen. Der Output gibt an, welche Funktionen überschrieben wurden.

3-2 Manipulierte Kernelmodule

Beschreibung: Kernelmodule können zu Laufzeit zusätzlich in den Kernel geladen werden. Die Manipulation erfolgt oft durch Rootkits, die sich im Kernel einlagern wollen.
autom. Datenerhebung: Ja.
autom. Auswertung: Nein.
Ansatz Datenerhebung: Kernelmodule befinden sich in den Unterverzeichnissen von */lib/modules* und besitzen die Extension „.ko“. Sie können allerdings auch von beliebigen Orten hinzugeladen werden. Eine Liste der aktuell geladenen Kernelmodule kann über das Modul „modules“ aus dem Volatility Framework geladen werden. Für das Auffinden von versteckten Kernelmodulen existiert das Modul *modscan*.
Ansatz Auswertung: Die Module können mit Tools wie „chrootkit“ und „Rootkit Hunter“ auf Rootkits gescannt werden. In der Ausgabe des Tools ist zu erkennen, ob das System mit einem Rootkit infiziert wurde, jedoch nicht inwiefern bestehende Kernelmodule manipuliert wurden.

4.1.5 Kategorie 4: Datei

Die Indikatoren dieser Kategorie behandeln Spuren an Dateien, die auf Malware hindeuten können.

4-1 Hashwert auf Blacklist

Beschreibung: Für Dateien bekannter Malware werden Hashwerte berechnet und als Blacklist in Datenbanken gespeichert. Diese werden z.B. von Malware Scannern genutzt, um Systeme auf eine Infizierung zu überprüfen.
autom. Datenerhebung: Ja.
autom. Auswertung: Ja.
Ansatz Datenerhebung: Für jede Datei, die im System überprüft werden soll, muss ein Hashwert berechnet werden. Die Hashfunktion muss mit der Funktion der Hashblacklist übereinstimmen. Ein Tool, mit dem der Hashwert (MD5) berechnet werden kann, ist *md5sum*.
Ansatz Auswertung: Die Hashwerte auf dem System müssen auf Überschneidungen mit der Blacklist geprüft werden. Finden sich Überschneidungen, deutet dies auf Malware hin. Dies kann von einem Malware Scanner, wie *Sophos* übernommen werden.

4-2 Malware Dateinamen

Beschreibung: Bekannte Malware kann anhand typischer Benennungen ihrer erstellten Dateien und Ordner erkannt werden. Teilweise erstellt Malware für ihre Dateien auch spezielle Dateiendungen.
autom. Datenerhebung: Ja.
autom. Auswertung: Ja.
Ansatz Datenerhebung: Um die Dateien auf dem System zu überprüfen müssen die zu untersuchenden Stellen gesichert werden. Hierfür wird entweder ein Abbild des gesamten Systems benötigt, oder die Überprüfung auf ausgewählte Pfade im System beschränkt.
Ansatz Auswertung: Für Dateinamen, die mit Malware in Verbindung stehen, können logische Ausdrücke formuliert werden. Mit dem Tool *Yara* kann ein System mit diesen logischen Ausdrücken überprüft werden.

4-3 Executables an ungewöhnlichen Orten

Beschreibung: Malware versucht die schädlichen Executables im System zu verstecken, indem Malware sie an ungewöhnlichen Orten, an denen normalerweise keine Executables liegen, ablegt.
autom. Datenerhebung: Ja.
autom. Auswertung: Nein.
Ansatz Datenerhebung: Das System muss an bestimmten Stellen nach Executabels abgesucht werden. Typische Orte sind */dev/shm*, */tmp*, */var/tmp*, */home*. Aber auch die Swappartition bzw. das Swapfile können als Versteck dienen. Mit dem Kommando *find <dir> -executable -type f* wird das System nach Dateien durchsucht, die ausführbar sind. Diese müssen in ihren Rechten als ausführbar markiert sein. Dateien, die ausführbaren code enthalten, aber selbst nicht die Rechte zur Ausführung besitzen, werden nicht berücksichtigt.
Ansatz Auswertung: Wurden Executables an ungewöhnlichen Orten gefunden, ist abzuklären, welche Funktion sie erfüllen. Die Überprüfung muss dabei manuell erfolgen.

4-4 Statisch gelinkte Dateien

Beschreibung: Statisch gelinkte Dateien werden oft von Malware Entwicklern verwendet, da statische Executables nicht von anderen Bibliotheken abhängig sind, sondern sämtlicher Code sich in der Executable befindet. Dies macht es zudem für Malware Analysten schwerer den Code zu interpretieren.
autom. Datenerhebung: Ja.
autom. Auswertung: Ja.
Ansatz Datenerhebung: Für die spätere Auswertung wird ein Abbild des Systems oder das Live-System benötigt, da alle Dateien überprüft werden sollen.
Ansatz Auswertung: Die Dateien können mit dem Befehl *ldd* auf Link-Abhängigkeiten

überprüft werden. Bei statisch gelinkten Dateien ist die Ausgabe des Befehls leer.

4-5 Content Extension Mismatch

Beschreibung: Der Dateityp einer Datei kann an einer Signatur im Header der Datei abgelesen werden. Ist die Datei anders benannt, ist es wahrscheinlich, das Malware bestimmte schädliche Dateien hinter harmlosen Dateinamen/-typen verstecken will.
autom. Datenerhebung: Ja.
autom. Auswertung: Ja.
Ansatz Datenerhebung: Für die spätere Auswertung wird ein Abbild des Systems oder das Live-System benötigt, da alle Dateien überprüft werden sollen.
Ansatz Auswertung: Mit dem Kommando *file* unter Linux können Informationen über eine Datei angezeigt werden. Die Forensik Software Autopsy beinhaltet das „File Extension Mismatch Module“, welches auf Übereinstimmung von Fileheader-Signatur und Dateiendung in der Benennung überprüft.

4-6 Linux untypische Extensions

Beschreibung: Linux-untypische Extensions können Spuren von Malware sein, die auf ein anderes Betriebssystem abzielt. Linux-Server werden häufig als C&C Server für Windows-Malware genutzt. Daher lassen sich auf dem System Windows-Binaries finden. Diese sind für das Linux System ungefährlich, können aber zur Weiterverbreitung von Malware beitragen.
autom. Datenerhebung: Ja.
autom. Auswertung: Ja.
Ansatz Datenerhebung: Für die spätere Auswertung wird ein Abbild des Systems oder das Live-System benötigt, da alle Dateien überprüft werden sollen.
Ansatz Auswertung: Für die Auswertung muss zuvor eine Liste mit Extensions erstellt werden, die für Linux untypisch sind. Mit Hilfe von regulären Ausrücken kann das System überprüft werden. Extensions, die sich nicht zuordnen lassen, müssen manuell überprüft werden.

4-7 UPX gepackte Executables

Beschreibung: Um Malware zu verstecken werden sogenannter Packer verwendet, um die Dateien zu komprimieren. Malware Executables sind häufig mit dem Packer UPX komprimiert. Dadurch kann Malware von Signatur-basierten Scannern nicht mehr gefunden werden. Die Executables werden dabei erst zu Laufzeit wieder dekomprimiert.
autom. Datenerhebung: Ja.
autom. Auswertung: Nein.

Ansatz Datenerhebung: Auf einem Abbild des Systems nach den Strings „UPX!“, „UPX0“, „UPX1“ und „UPX2“ suchen.
Ansatz Auswertung: Wenn eine mit UPX komprimierte Datei gefunden ist, kann diese wieder dekomprimiert werden. Anschließend muss eine manuelle Analyse erfolgen, um die Funktion der Datei festzustellen.

4-8 Applikationen legen Dateien an ungewöhnlichen Orten ab

Beschreibung: Bestimmte Bereiche des Betriebssystems werden bevorzugt als Ablageort für Dateien von Programmen genutzt. Andere Bereiche dienen der Konfiguration des System und sollten nur vom Administrator modifiziert werden. Wenn Malware versucht, das System zu manipulieren, kann es vorkommen, das Dateien an ungewöhnlichen Orten abgelegt werden. Zu den ungewöhnlichen Orten zählen: */etc*, */usr*, */lib*, */boot*, */dev*, */sbin*, */srv*.
autom. Datenerhebung: Nein.
autom. Auswertung: Nein.
Ansatz Datenerhebung: Mit *find* können die innerhalb eines festgelegten Zeitraums veränderten Dateien angezeigt werden. Eine Suche nach kürzlich erstellten Dateien ist nicht möglich, da Dateien keinen Zeitstempel für ihr Erstellung besitzen. Einen direkten Weg, um von Applikationen erstellte Dateien zu finden, gibt es nicht. Über den Befehl *lsof* kann angezeigt werden, welcher Prozess die Datei erstellt hat, wenn dieser noch existiert.
Ansatz Auswertung: Wenn alle kürzlich erstellten Dateien im System gefunden wurden, muss der Dateipfad geprüft werden. Sollte dieser innerhalb der in der Beschreibung genannten Pfade liegen muss geprüft werden, wie die Datei erstellt wurde und welchen Inhalt sie hat. Da aber keine eindeutige Verbindung zwischen einer Datei und einem Programm bzw. Prozess hergestellt werden kann, kann es sich bei geänderten Dateien innerhalb der Pfade auch um legitime Änderungen des Systems handeln.

4.1.6 Katgeorie 5: Netzwerk

Indikatoren dieser Kategorie behandeln Malware-Spuren, die im Zusammenhang mit Netzwerkverkehr auftreten.

5-1 Ungewöhnliche Ports/ Dienste

Beschreibung: Bestimmte Dienste laufen über standardisierte Ports. Laufen diese Dienste über andere Ports oder andere Dienste als erwartet auf einem Port, kann dies auf Malware hindeuten. Diese benötigt für die Kommunikation mit dem C&C Server, Kanäle um nach außen zu kommunizieren.

autom. Datenerhebung: Ja.
autom. Auswertung: Nein.
Ansatz Datenerhebung: Alle Netzwerkverbindungen können auf dem System über das Tool „netstat“ erhoben werden. Wenn nur ein RAM-Image vorliegt, gelangt man mit dem Modul *linux_netstat* vom Volatility Framework an alle aktiven Netzwerkverbindungen.
Ansatz Auswertung: Von den Systemverantwortlichen sollte in Erfahrung gebracht werden, welche Dienste auf welchen Ports für das System vorgesehen sind. Die Informationen müssen mit den erhobenen Daten abgeglichen werden.

5-2 Malware IP-Adressen

Beschreibung: Für IP Adressen, die mit Malware in Verbindung stehen, existieren ebenfalls Blacklists. Bestimmte Adressen können zum Nachladen von Malware dienen. Verbindet sich ein System mit diesen Adressen, ist es bereits infiziert, da das Verbinden zu bestimmten IP-Adressen bereits in einem vorhergehenden Malware-Skript initiiert wurde.
autom. Datenerhebung: Ja.
autom. Auswertung: Ja.
Ansatz Datenerhebung: Die Netzwerkverbindungen können mit *netstat* oder mit dem *linux_netstat* Modul aus dem Volatility Framework angezeigt werden. Im Feld Destination IP sind die Ziel-IP-Adressen der Verbindungen aufgelistet.
Ansatz Auswertung: Die Liste der IP-Adressen, zu denen sich das System verbunden hat, kann mit der Blacklist von schädlichen IP-Adressen abgeglichen werden. Bei Übereinstimmungen wird dies als Indikator für Malware gewertet.

5-3 Malware Domain Names

Beschreibung: Für Domain Names, die mit C&C Servern von Malware in Verbindung stehen, existieren Blacklists. Verbindet sich das System mit einer Adresse eines C&C Servers, ist davon auszugehen, dass das System bereits infiziert ist. C&C Server dienen Malware Entwicklern zur Koordination von gezielten Angriffen.
autom. Datenerhebung: Nein.
autom. Auswertung: Ja.
Ansatz Datenerhebung: Die Domain Names der letzten Verbindungen können aus dem DNS Cache extrahiert werden. Linux besitzt standardmäßig keinen DNS Cache auf Ebene des Betriebssystems. Dieser ist daher, wenn er vorhanden ist, auf Ebene des Browsers zu finden.
Ansatz Auswertung: Die im DNS Cache gefundenen Domain Names werden mit der Blacklist aus bekannten schädlichen Domain Names verglichen. Bei Übereinstimmungen wird der Indikator als erfüllt angesehen.

4.1.7 Kategorie 6: Pakete

Indikatoren dieser Kategorie beschreiben Malware-Spuren, die sich auf Pakete beziehen, die über einen Paketmanager auf dem System installiert werden können.

6-1 Nicht verifizierte Pakete

Beschreibung: Pakete, die über einen Paketmanager heruntergeladen wurden, lassen sich anhand des Hashwertes auf Manipulationen überprüfen. Malware kann versuchen harmlose Pakete auf dem System zu manipulieren, um schädliche Aktivitäten im System zu verstecken.
autom. Datenerhebung: Ja.
autom. Auswertung: Ja.
Ansatz Datenerhebung: Für die Auswertung wird ein Abbild des Systems oder Live-System benötigt. Der Paketmanager *apt* legt die installierten Pakete unter */var/cache/apt/archives* ab. Unter Red Hat Enterprise Linux werden die Pakete mit *yum* verwaltet und unter */var/cache/yum/* abgelegt.
Ansatz Auswertung: Viele Paketmanager bringen bereits eine Funktion zum verifizieren der installierten Pakete mit. Der Redhat Paketmanager setzt dies mit dem Kommando *rpm -V* um. Der Debian Package Manager liefert mit *dbkg -V* ebenfalls eine Option zur Hashprüfung von installierten Paketen. Aus der Ausgabe ist zu erkennen, ob und welche Pakete manipuliert wurden.

6-2 Nicht vorgesehene Pakete

Beschreibung: Bei Paketen, die auf dem System nicht vorgesehen sind, kann es sich um Pakete handeln, die von Malware zusätzlich hinzugefügt wurden. Über zusätzlich installierte Pakete soll Malware möglichst unauffällig im System installiert werden.
autom. Datenerhebung: Nein.
autom. Auswertung: Nein.
Ansatz Datenerhebung: Über einen Paketmanager lassen sich alle über den Paketmanager installieren Pakete auf dem System anzeigen. Pakete die aus anderen Quellen zu dem System hinzugefügt wurden, können so nicht gefunden werden. Zusätzlich kann die *PATH* Variable ausgelesen werden, um Executables von Programmen auf dem System aufzulisten. Unter Linux kann dies über das Kommando *compgen -c* umgesetzt werden.
Ansatz Auswertung: Zusammen mit den Systemverantwortlichen sollte die Liste der installierten Pakete auf ungewollte Pakete überprüft werden.

4.2 Test der Automatisierbarkeit

Um die Automatisierbarkeit auf Linux zu testen, sollen einzelne IoCs in einem Skript umgesetzt werden, das ein forensisches Image auf Spuren von Malware überprüft.

Für den Versuch werden zwei Opfersysteme in Form von virtuellen Maschinen aufgesetzt. Diese sollen jeweils mit Malware infiziert werden, indem die schädliche Datei mit Root-Rechten auf dem System ausgeführt wird. Eine weitere VM soll für die Analyse bereit stehen.

Das Skript zur Datenerhebung wird über einen gemeinsamen Ordner in die Opfersysteme gemounted. Anschließend muss das Skript mit Root-Rechten ausgeführt werden. Die gesammelten Daten werden im gemeinsamen Ordner abgelegt. Dabei darf nur lesend auf die Daten zugegriffen werden, um Manipulationen zu vermeiden. Anschließend kann der Ordner aus dem System ausgehangen werden.

Der Ordner mit den gesammelten Daten kann nun an das Analysesystem gemounted werden. Die Skripte zur Vorverarbeitung und Analyse der Daten befinden sich bereits auf dem Analysesystem und werden dort ausgeführt. Die Ergebnisse werden abschließend dahingehend geprüft, ob die Malware erkannt wurde.

4.2.1 Konzeptionelle Entscheidungen

In den folgenden Abschnitten werden Entscheidungen, die den Inhalt des Test betreffen, aufgeführt. Dies Betrifft den Aufbau des Skriptes, die Auswahl der IoCs aber auch die ausgewählte Malware.

Grundstruktur des Skripts

Das Skript setzt sich aus zwei Teilen zusammen:

- Datenerhebung
- Auswertung der Daten

Der erste Teil des Skriptes ist ein Shell-Skript, welches für die automatische Erhebung der Daten zuständig ist (siehe Abb.4.2). Das Skript sichert zunächst die flüchtigen Daten in Form eines RAM-Image. Dies wird mit dem Tool Linux Memory Extractor[1] (LiME) umgesetzt. Hieraus können später die aktiven Netzwerkverbindungen, geladene Kernelmodule und aktive Prozesse extrahiert werden.

[1] GitHub-Repository von LiME: https://github.com/504ensicsLabs/LiME

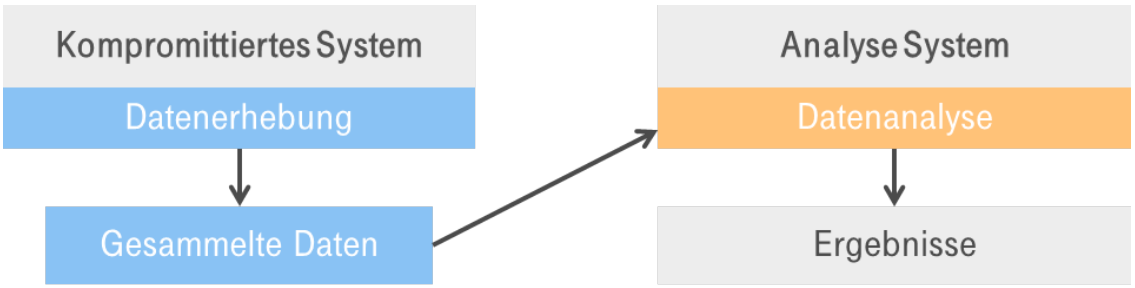

Abbildung 4.1: Skriptaufteilung auf zu untersuchendes kompromittiertes System und Analysesystem (Quelle: eigene Darstellung)

Anschließend werden weitere Daten erhoben und als Text-Datei gespeichert. Dies betrifft:

- */etc/passwd*
- */etc/shadow*
- */etc/group*

Nach Sicherung der wichtigsten Daten, wird auf dem System ein Malware Scan durchgeführt. Die vom Scanner erzeugten Log-Dateien werden ebenfalls gesichert.

Bei dem Analyse-Skript (siehe Abb.: 4.3) handelt es sich um ein Bash-Skript, welches per Kommandozeilenbefehl auf dem Analyse-System gestartet werden kann. Als Parameter wird der Pfad zum RAM Image übergeben. Die Aufgabe des Skriptes ist es, die nötigen Tools zu starten, um die vorliegenden Daten zu analysieren. Die Ergebnisse werden als Text-Datei ausgegeben.

Wesentlicher Bestandteil des Analyse Skriptes ist das Volatility Framework[4], welches verschiedene Module zum analysieren von RAM-Images bereitstellt. Um das RAM-Image zu analysieren, benötigt Volatility ein Profil, dass Informationen über die Struktur des Kernels enthält. Für Windows existieren bereits viele vorgefertigte Profile, für Linux müssen diese selber erstellt werden. Dieser Schritt erfolgt manuell und ist nicht Teil des Skriptes. Im GitHub-Repository findet sich eine Anleitung für das Erstellen von Volatility-Profilen[5].

Innerhalb des Skriptes werden die Module von Volatility wie folgt aufgerufen:

```
python vol.py -f <RAM-Image> --profil=<Profil> <Modul>
```

Mit Hilfe von Python wird Volatility gestartet und als erster Parameter der Pfad zum

[4] GitHub-Repository von Volatility: https://github.com/volatilityfoundation/volatility
[5] Anleitung zum Erstellen neuer Profile: https://github.com/volatilityfoundation/volatility/wiki/Linux#creating-a-new-profile

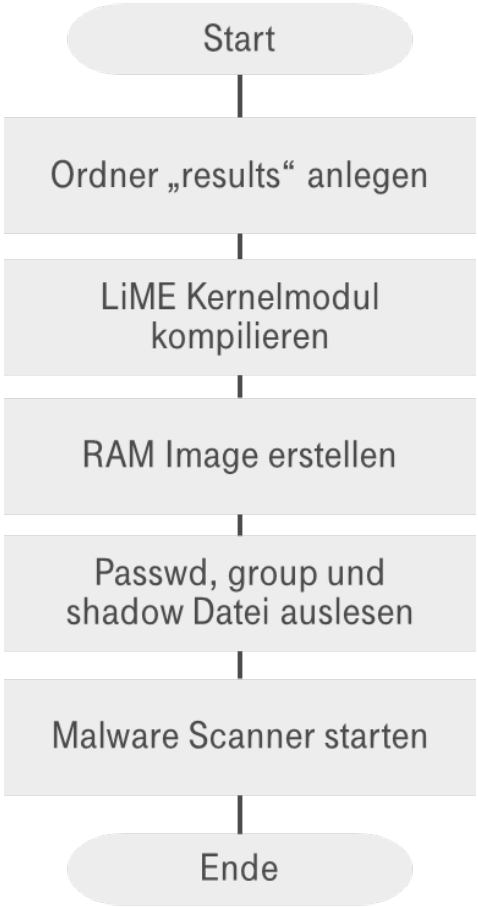

Abbildung 4.2: Ablaufplan des Bash-Skriptes „collect.sh“ (Quelle: eigene Darstellung)

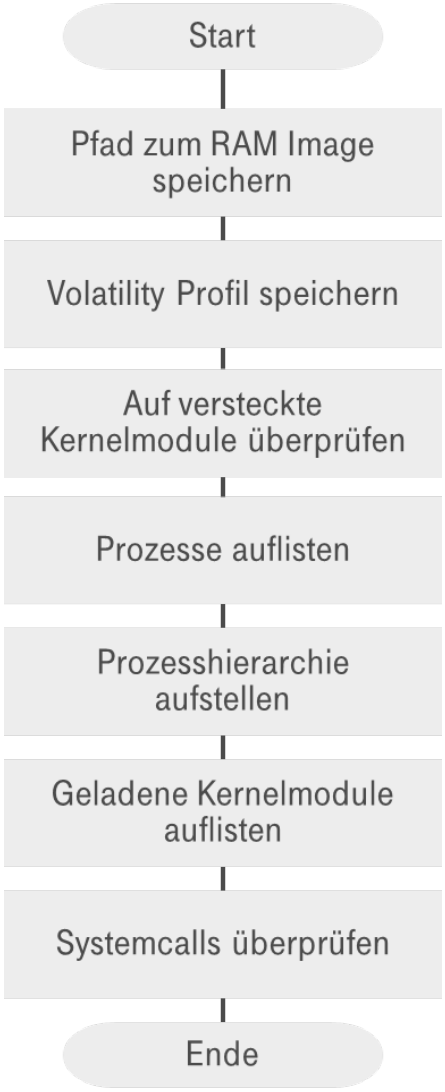

Abbildung 4.3: Ablaufplan des Bash-Skriptes „analyse.sh“ (Quelle: eigene Darstellung)

RAM-Image übergeben. Anschließend wird das ermittelte Profil übergeben und am Ende das gewünschte Modul aufgerufen.

Auswahl der IoCs

Für den Test sollen nicht alle aufgestellten IoCs in einem Skript umgesetzt werden, sondern nur ausgewählte Indikatoren.

Ein Indikator, der möglichst viel Malware verschiedenster Klassen findet, ist die Überprüfung auf Hashwerte, die mit Malware in Verbindung stehen (sogenannte Virensignaturen). Anbieter von Antivirensoftware pflegen Datenbanken mit Hashwerten für bekannte Malware. Mit diesem Indikator kann Malware, insofern sie bekannt ist, zuverlässig gefunden werden.

Um auch Malware zu finden, die selbstständig ihre Dateien verändert, um eine Signaturüberprüfung zu umgehen, müssen Indikatoren verwendet werden, die auf das Verhalten von Malware abzielen. So zum Beispiel das Erstellen und Verändern von Nutzern auf dem System bei der Übernahme von Accounts und der Rechteerweiterung.

Ein weiterer Indikator für einen Malwarebefall sind manipulierte Systemcalls. Dieses Vorgehen wird bevorzugt von Rootkits auf Linux verwendet. Daher zielt der Indikator speziell auf diese Klasse von Malware ab.

Malware die sich mit Root-Rechten im System ausbreitet ist potenziell in der Lage Kernelmodule nachzuladen oder zu verstecken. Daher wird dies im Skript ebenfalls überprüft.

Das Ausführen von Malware führt zwangsläufig zum Aufruf eines neuen Prozesses. Die Überprüfung der Prozesse wird eingeführt, um Malware, unabhängig von ihrer speziellen Funktion, zu finden.

Umsetzung der Überprüfung im Skript

In den folgenden Abschnitten wird jeweils ein kurzer Überblick zu der Umsetzung der Überprüfung, wie sie innerhalb des Skriptes erfolgt, gegeben.

Überprüfung der Prozesse
Im Analyse Skript werden die aktiven Prozesse der virtuellen Maschine aus dem RAM-Image extrahiert. Hierfür wird das Modul *linux_pslist* eingesetzt. Des weiteren wird das Modul *linux_pidhashtable* eingesetzt. Die Module bauen beide eine Übersicht der aktiven Prozesse auf, benutzten aber unterschiedliche Methoden. Die Ausgaben der Module werden für die Analyse miteinander verglichen. Abweichungen können versteckte Prozesse erkennbar machen, dies muss jedoch manuell nachgeprüft werden.

Zusätzlich können die Beziehungen zwischen Prozessen rekonstruiert werden. Dafür wird das Modul *linux_pstree* von Volatility aufgerufen. Die Ausgabe des Befehls erzeugt eine Übersicht aus aktiven Prozessen und ordnet Kindprozesse ihren Elternprozessen zu.

Signaturüberprüfung
Im Teil des Skriptes, das auf dem kompromittiertem System die Datenerhebung durchführt, wird nach dem Sammeln der Daten ein Malwarescan durchgeführt. Die Aufgabe der Signaturüberprüfung wird von dem Tool *Sophos Anti-Virus*[6] umgesetzt. *Sophos Anti-Virus* wurde ausgewählte, da es kostenlos zur Verfügung steht und laut *av-test* über eine hohe Erkennungsrate von 95% bei Linux-Malware verfügt [Sel15]. Zusätzlich wird mit dem Tool *rkhunter*[7] speziell auf Rootkits und Exploits im System gescannt. Dieses wird nach der Erzeugung des RAM-Images und Erhebung der restlichen Daten gestartet.

Überprüfung der User
Um die User auf dem kompromittiertem System überprüfen zu können wird im ersten Teil der Datenerhebung die passwd-Datei gesichert. In dieser sind alle auf dem System vorkommenden User aufgelistet. Die Überprüfung kann im Anschluss durch einen Abgleich mit einer Whitelist umgesetzt werden. Dies wird in diesem Skript durch einen Abgleich mit der /etc/passwd aus einem früheren Zustand der VM ersetzt.

Überprüfung der Systemcalls
Als Datengrundlage für die Überprüfung der Systemcalls des Systems dient das erzeugte RAM-Image. Für die Auswertung wird das Modul *linux_syscall_check* des Volatility Frameworks gestartet. Dieses überprüft die Systemcall Tabelle aus dem RAM-Image auf manipulierte Pointer, die nicht auf Code im Kernel zeigen. In der Ausgabe des Tools finden sich, bei manipulierten Systemcalls, in der Ausgabe als Vermerk das Wort „HOOKED“. Nach diesem wird mit Hilfe von *grep* gesucht.

Überprüfung der Kernelmodule
Durch das zweite Modul *linux_check_inline_kernel* wird zudem der Code innerhalb des Kernels auf Veränderungen überprüft. Das Modul *linux_check_moduls* überprüft die Liste der geladenen Kernelmodule auf versteckte Module, da Rootkits häufig ihre Aktivität verschleiern. Um einen Überblick über alle geladenen Kernelmodule zu bekommen wird zudem das Volatility Modul *linux_lsmod* eingesetzt, welches die geladenen Kernelmodule auflistet.

[6] Website Sophos Anti-Virus für Linux : https://www.sophos.com/de-de/products/free-tools/sophos-antivirus-for-linux.aspx
[7] Rootkit Hunter Projekt: http://rkhunter.sourceforge.net/

Auswahl der Malware

Für die Beschaffung von Malware Samples wurden folgende Datenbanken durchsucht (Stand: 12.08.2017):

- contagio malware dump[8]: 6 Ergebnisse die auf Label „Linux“ matchen
- theZoo[9]: 3 Dateien mit Bezeichnung „Linux“
- kernelmode.info[10]: 42 Einträge für Linux-basierte Malware
- avcaesar.malware[11]: 927 Ergebnisse für den Begriff „Linux“
- malwr.com[12]: 3 Ergebnisse bei Suche nach „Linux“ Tag
- VirusShare: Mehr als 20 Ergebnisse (genaue Anzahl kann nicht dargestellt werden)

Viele der Ergebnisse mussten allerdings verworfen werden, da sich die Malware auf die falsche Architektur bezog. Unter den Beispielen war häufig Malware für Android zu finden, welche für den Test ausgeschlossen wurde.

Als weitere Quelle für Malware Samples wurde GitHub[13] durchsucht. Dabei konnten an folgenden Stellen Quellcode für Linux Malware gefunden werden (Stand: 12.08.2017):

- gonnaCry[14]: Ransomware für Linux
- Phalanx[15]: Rootkit
- Skeksi[16]: Virus
- nicht näher bezeichnetes Rootkit[17]

Die Suche nach spezifischer Malware (killDisk, Ebury) ergab keine Treffer. Für die Malware „Turla“ konnte ein Beispiel auf der Seite VirusShare gefunden werden.

Für den Test wurden 4 Malware Beispiele auf der Ubuntu VM installiert:

- **gonnaCry**: Bei gonnaCry handelt es sich um eine, für Testzwecke, entwickelte Ransomware für Linux, d.h. sie ist in der Lage Dateien zu verschlüsseln. Sie verfügt über eine Testfunktion, welche Testdaten verschlüsselt, so dass nur ein Teil des Systems verschlüsselt wird und der Zugriff auf den Rest des System weiterhin gewährt wird.
- **Phalanx**: Ist ein Rootkit für Linux, dessen Quellcode 2005 veröffentlicht wurde.

[8] Link zu den Suchergebnissen: http://contagiodump.blogspot.de/search/label/Linux
[9] Link zu den Suchergebnissen: https://github.com/ytisf/theZoo/tree/master/malwares/Binaries
[10] Link zu den Suchergebnissen: http://www.kernelmode.info/forum/viewtopic.php?f=16&t=3471
[11] Link zu den Suchergebnissen: https://avcaesar.malware.lu/sample/search?query=Linux
[12] Link zu den Suchergebnissen: https://malwr.com/analysis/search/
[13] GitHub ist eine Plattform für die Verwaltung von Software-Entwicklungsprojekten: https://github.com/
[14] GitHub Repository: https://github.com/tarcisio-marinho/GonnaCry
[15] GitHub Repository: https://github.com/unixfreaxjp/LinuxMalwareSourceCode/tree/master/LKMRootkits
[16] GitHub Repository: https://github.com/unixfreaxjp/LinuxMalwareSourceCode/tree/master/Virus
[17] GitHub Repository: https://github.com/ivyl/rootkit

- **Skeksi**: Der Virus befällt dynamisch gelinkte ELF-Binaries und verändert Abschnitte innerhalb der Datei, um Platz für beliebigen Schadcode zu schaffen. Wird die Datei nach der Infizierung ausgeführt, kommt der injizierte Code zur Ausführung.
- **Turla**: Turla ist ein Trojaner für Linux, der ursprünglich für Windows entwickelt wurde. 2014 tauchte erstmals die Linux Variante auf. Er verfügt über sehr fortgeschrittene Techniken, das eigene Verhalten zu verschleiern.

Andere Malware Beispiele wurden ebenfalls auf dem Testsystem installiert, konnten aber nicht fehlerfrei zur Ausführung gebracht werden und mussten daher verworfen werden.

Virtuelle Maschinen

Als Opfersysteme wird eine virtuelle Maschinen aufgesetzt. Sie basiert auf einem Ubuntu der Version 17.04. Diese wurde ausgewählt, da die meisten Webserver auf Ubuntu und Debian basieren [w3t17].
Von jeder Infizierung mit einer Malware wird ein Snapshot erstellt, der später vom Datensammelskript untersucht wird.

Eine weitere VM wird aufgesetzt. Sie stellt das System dar, auf dem die gesammelten Daten analysiert werden. In diesem Testfall basiert die Maschine ebenfalls auf einem Ubuntu, da die Analyse Skripte als Bash Skript vorliegen. Das Betriebssystem der Analyse-VM spielt für die Prüfung der Durchführbarkeit einer automatisierten Malwareüberprüfung auf Linux keine Rolle.

Die Virtualisierung wird mit Hilfe von *Oracle VM Virtual Box*[18] realisiert.

4.2.2 Testergebnisse

Mit Hilfe der Skripte konnte bei jeder Infizierung der VM ein Abbild des Hauptspeichers erzeugt werden. Dieses wurde durch die vom Volatility Framework bereitgestellten Module untersucht. Die Ergebnisse wurden als Text-Dateien gespeichert, die anschließend manuell weiter ausgewertet werden können.

[18] Link zur Seite: https://www.virtualbox.org/

Bei einer ersten manuellen Überprüfung der Ergebnis-Daten konnten bisher keine Indikatoren für Malware Aktivitäten gefunden werden.

- Die Überprüfung der Prozesse ergab Unterschiede zwischen dem Output von *linux_pslist* und *linux_pidhashtable*. Bei einer manuellen Überprüfung der Prozesse konnten die Unterschiede jedoch nicht eindeutig einem schädlichen Prozess zugeordnet werden.
- Es wurden keine Veränderungen bei den Usern des Systems festgestellt.
- Die Überprüfung der Systemcalls war in der Skriptumsetzung fehlerhaft. Daher konnten keine Ergebnisse erzeugt werden.
- Es wurden keine versteckten oder manipulierten Kernelmodule gefunden.

Der Output des Malware Scanners war der einzige Hinweis auf die Existenz von Malware. Durch den Scanner konnten so Hinweise auf Turla und Phalanx gefunden werden.

5 Diskussion

Das folgende Kapitel dient der Reflexion der Arbeit und der Bildung eines Fazits. Des weiteren wird ein Ausblick für zukünftige Ansätze und Problemstellungen bei der Erkennung von Malware in Linux-basierten Systemen gegeben.

5.1 Einschränkungen der Automatisierbarkeit

Ein grundlegendes Problem bei der automatisierten Suche nach Malwarespuren mit Hilfe von IoCs ist die freie Definition des Begriffs „Indicator of Compromise“. Indikatoren für eine Infizierung mit Malware können sowohl sehr komplex als auch trivial sein. Alle Indikatoren müssen, um in eine automatisierte Lösung integriert werden zu können, maschinenlesbar beschrieben werden. Dies ist aufgrund der freien Auslegung des Begriffs erschwert.

Ein weiteres Problem entsteht durch das Fehlen eines standardisierten Formates zur Dokumentation von IoCs. Dies behindert den Austausch von gefundenen IoCs zwischen verschiedenen Forschungsinstituten in der Malware Forschung und verhindert das Entstehen einer Community, die den öffentlichen Austausch von IoCs voran treibt. Somit gibt es keine gepflegte Plattform, die IoCs sammelt, auf die ein Tool zugreifen könnte.

Bei Linux ergeben sich, aufgrund der Unterschiede zwischen den einzelnen Distributionen, bereits Probleme bei der Automatisierung der Datenerhebung. Indikatoren, die sich auf den Kernel beziehen sind davon weniger betroffen. Bei anderen Indikatoren, die sich auf Software bezieht treten Probleme aufgrund der unterschiedlichen Standards zwischen den Distributionen auf. So z.B. bei standardmäßig installierten Paketverwaltung. Je nachdem, welcher Paketmanager verwendet wird, unterscheiden sich die Ablageorte der installierten Pakete bzw. stehen unterschiedliche Funktionen zur Verifizierung dieser zur Verfügung.

Eine weitere Einschränkung entsteht bei der Erstellung eines RAM-Image. Eine automatische Erhebung eines Hauptspeicherabbilds ist schwierig umzusetzen, da der Linux Kernel keinen direkten Zugriff auf */dev/mem* zulässt. Für die Erhebung des RAM-Images muss daher das Programm LiME gestartet werden, welches ein Kernelmodul erzeugt, mit Hilfe dessen das Speicherabbild erhoben werden kann. Dieses muss in das zu untersuchende System eingebunden werden. Bei der automatisierten Lösung muss das Kompilieren des Kernelmoduls auf dem zu untersuchendem System stattfinden. Daher wird das Arbeitsspeicherabbild verfälscht. Um ein unverfälschtes Abbild zu bekommen, muss das Kernelmodul zuvor auf einem anderen System mittels Cross-Compiling erstellt werden. Dies muss manuell umgesetzt werden und steht entgegen einer vollautomatisierten

Lösung.

5.2 Automatisierbarkeit im Vergleich zu Windows

Das erstellte Skript zur automatisierten Datenerhebung und Analyse orientiert sich im Aufbau und erhobenen Daten stark an dem für Windows Systeme entwickeltem Skript.

In beiden Fällen wird ein RAM-Image erhoben, das in der späteren Analyse mit Volatility ausgewertet wird. Die Module, die bei der Auswertung aufgerufen werden unterscheiden sich. Bei dem für Windows entwickeltem Skript werden z.B. die aktiven Netzwerkverbindungen aus dem RAM-Image ausgelesen. Bei dem Skript für Linux wurde darauf vorerst verzichtet.Für Linux existiert ebenfalls ein Volatility Modul, mit dem Netzwerkverbindungen aus dem RAM-Image ausgelesen werden können. Daher kann diese Funktion nachträglich ebenfalls eingebaut werden.

Bei den zusätzlich zum RAM-Image erhobenen Daten unterscheiden sich die Skripte. Bei Windows werden nur die Autostarts zusätzlich erhoben. Bei Linux findet zudem eine Sicherung der auf dem System existierenden User und Gruppen statt. Hier ist die bei Linux implementierte Lösung weitreichender, da Manipulationen der User ebenfalls erkannt werden können.

Die Windows Skripte basieren hauptsächlich auf der Untersuchung des Hauptspeichers und der Autostartregion. Der Vorteil dieses Vorgehens ist eine schnelle Datenerhebung, da kein komplettes Image sondern nur ein Triage Image nötig ist. Bei dem Ansatz für Linux werden mehr Daten, als ein Hauptspeicherabbild erhoben, um Aussagen über eine mögliche Malwareinfizierung des Systems treffen zu können. Die Betrachtung des Systems ist daher umfangreicher und kann potenziell mehr Malware erkennen. Ein Nachteil einer umfangreicheren Untersuchung des Systems ist der erhöhte Zeitaufwand.

5.3 Beurteilung des Tests

Mit dem Testszenario konnte eindeutig gezeigt werden, das die automatisierte Datenerhebung für die ausgewählten IoCs realisierbar ist. Somit wurden die Grundlagen für eine anschließende Analyse gelegt. Die Umsetzung einer vollautomatisierten Analyse ist jedoch schwierig, da innerhalb des Skriptes noch keine vollautomatisierte Überprüfung stattfindet, ob Malware-Spuren vorliegen. Es ist bisher immer eine manuelle Überprüfung nötig. Hierfür müssten erst Regeln aufgestellt werden, nach denen der Output der Volatility Module auf schädliche Artefakte durchsucht wird. Dies war innerhalb der zeitlichen Begrenzung der Bachelorarbeit nicht möglich.

Des weiteren konnten die ausgewählten Indikatoren nicht optimal überprüft werden, da

teilweise die Auswirkungen der zu testen Malware auf das System unbekannt waren. Theoretisch kann die Malware ein Verhalten gezeigt haben, dass nicht von den Indikatoren erfasst wurde. Alternativ wäre eine Entwicklung von eigener Malware zu Testzwecken mit vordefinierter Funktion möglich gewesen. Diese Ansätze mussten jedoch aufgrund der begrenzten Zeit für die Arbeit ausgeschlossen werden. Zudem wäre der Test unter verfälschten Bedingungen abgelaufen. Durch die Auswahl von Live-Beispielen von Malware, wie Turla, wurde eine reale Testumgebung geschaffen.

Bei der Auswahl der Malware für das Testszenario traten Probleme aufgrund der Verfügbarkeit von Malware Samples auf. Die gängigen Malwaredatenbanken,
wie z.B. *www.malwr.com* bieten meist nur eine Suche nach Malware-Hashes an, um bestimmte Einträge zu finden. Die meisten Datenbanken scheinen auch keine Linux-Malware zu führen oder lassen sich aufgrund der Größe und schlechter Suchalgorithmen nur sehr schwer durchsuchen. Eine Filterung nach Betriebssystem- oder Architekturspezifischer Malware war nicht möglich. Daher musste teilweise auf einzelne Veröffentlichungen von Malware Code auf GitHub zurückgegriffen werden.

Bei der auf GitHub veröffentlichten Malware handelt es sich meist um, für wissenschaftliche Zwecke entwickelten Code ohne die Absicht, diesen für kriminelle Zwecke zu verbreiten. Daher ist fraglich, ob die Malware Entwickler die gleichen Standards bei der Entwicklung des Codes verwendet haben, wie es Malware Entwickler mit krimineller Absicht tun. Teilweise konnten die gefundenen Beispiele nicht lauffähig gemacht werden. Nicht immer war die Funktion der Malware dokumentiert, so dass eine schädliche Absicht nicht immer bestätigt werden konnte.

Bei der Erstellung des Skriptes gab es Probleme bei dem Aufruf des Volatility Moduls *linux_check_syscall*, da dies nicht fehlerfrei ausgeführt wurde. Diese Fehlfunktion wurde während des Tests aufgedeckt und muss in Zukunft behoben werden.

Ein weiteres Problem ist, dass der Aufruf des Malware und Rootkit Scanners zwar innerhalb des Skriptes implementiert ist, jedoch eine Installation der Tools auf dem Opfersystem voraussetzt. Daher kann das Skript nicht beliebig auf ein frei gewähltes System eingesetzt werden. Bisher wurde noch kein Malware Scanner für Linux gefunden, der nicht notwendiger weise auf dem Zielsystem installiert werden muss.

Die wichtigste Erkenntnis aus dem Test war, das trotz umfangreicher IoCs nicht jedes Verhalten von Malware abgefangen wird. Malware die Bereiche innerhalb von Dateien verändert, wurde im Test nicht erkannt, da das Skript dieses Verhalten nicht abdeckte. Malware verfügt zudem über fortgeschrittene Taktiken zur Verschleierung von schädlichen Prozessen, so dass die Veränderungen des Systems nicht auffällt. Vermutlich konnten daher aus der Überprüfung der Prozesse keine Erkenntnisse gezogen werden.

Der Test war wichtig, um einschätzen zu können, wie herausfordernd die Erkennung

von Malware sein kann. Der Test zeigt auf, das ein Skript, welches zur zuverlässigen Erkennung von Malware genutzt werden soll, einen weitaus umfangreicheren Entwicklungsprozess bedarf, als er in dieser Bachelorarbeit gegeben werden kann.

5.4 Gefundene IoCs

Innerhalb der Arbeit wurde eine Übersicht mit IoCs für Linux zusammengetragen. Diese inhaltlich verschiedene Schwerpunkte und decken somit einen breiten Bereich an denkbarer Malware ab. Die vom Autor erstelle Übersicht ist jedoch keine vollständige Liste. Die aufgeführten IoCs sind sehr allgemein gehalten, was das Entstehen von falschpositiven Treffern theoretisch begünstigt. Weitere Indikatoren sind denkbar.

Aus dem Ergebnis des Tests kann abgeleitet werden, dass Malware unter Linux bereits auf einem sehr hohem Niveau entwickelt wird und höhere Techniken zur Verschleierung der Aktivität nutzt. Daher müssen die Indikatoren weiterentwickelt werden, um auch diese Malware auf dem System erkennen zu können. Es kann nicht ausgeschlossen werden, dass für eine effektive Malware Erkennung komplexere Indikatoren nötig sind.

5.5 Schlussbetrachtung und Ausblick

Ziel der Arbeit war es, eine Vorüberlegung zu Treffen, ob eine automatisierte oder zumindest teil-automatisierte Erkennung von Linux Malware mit Hilfe von IoCs möglich ist. Dafür wurden vom Autor eigene IoCs aufgestellt, die sich möglichst allgemein für die Erkennung von Malware eignen. Durch den Entwurf von Ansätzen für Skripte und die Tests mit kompromittierten virtuellen Maschinen konnte gezeigt werden, dass eine teil-automatisierte Erkennung von Malware innerhalb von Linux Systemen möglich ist. Automatisiert kann vor allem die Datenerhebung durchgeführt werden. Die Automatisierung der Auswertung der Daten konnte im Test bisher nicht umgesetzt werden. Die theoretischen Ansätze wurden vom Autor bereits aufgestellt.

Die Arbeit legte die theoretischen Grundlagen, um zukünftig Software zu entwickeln, die eine Überprüfung von Linux Systemen auf Malware durchführt. Zuvor müssen die gefundenen IoCs jedoch mit weiteren Testdaten evaluiert werden. Es muss geprüft werden, welches Malware-Verhalten durch die IoCs noch nicht abgebildet wird und an welchen Stellen die Indikatoren nach geschärft werden müssen, um eine möglichst geringe falschpositiv Rate zu erhalten. Hierfür bietet es sich an, eigene Malware für den Testzweck zu entwickeln.

In weitere Test muss überprüft werden, inwiefern die gefundenen IoCs für alle Distributionen von Linux zutreffen, da gerade Dateipfade auf denen einige Indikatoren basieren, sich unterscheiden. Interessant ist auch, inwiefern sich Linux-basierte Betriebssysteme,

wie Android, für mobile Geräte mit den IoCs untersuchen lassen. Da Android bei Smartphones am weitesten verbreitet ist, existiert hierfür bereits wesentlich mehr Malware, als für Server oder Desktop Systeme. Dies veranlasst wiederum mehr Forschungsteams im Bereich der Malware Erkennung auf Android zu forschen, so dass in diesem Bereich bereits mehr Wissen zusammen getragen wurde.

Da die in der Arbeit aufgestellte Übersicht über Indikatoren keine vollständige Liste ist, könnte sich eine zukünftige Arbeit mit der Aufstellung von weiteren Indikatoren beschäftigen. Interessant wäre ein Ansatz über maschinelles Lernen, so dass die Indikatoren auf Basis von Testdatensätzen gebildet werden. Denkbar wäre es, speziellere Indikatoren aufzustellen, die sich auf eine bestimmte Malware-Familie beziehen.

Literaturverzeichnis

[Arg17] ARGHIRE, I.: *KillDisk Malware Targets Linux Machines.* http://www.securityweek.com/killdisk-malware-targets-linux-machines. Version: 2017, Abruf: 28.03.2017

[Asc14] ASCHERMANN, T.: *Was ist Malware?* http://praxistipps.chip.de/was-ist-malware_28542. Version: 2014, Abruf: 27.03.2017

[Ben17] BENZMÜLLER, Ralf: *Malware-Trends 2017.* https://www.gdata.de/blog/2017/04/29667-malware-trends-2017. Version: 2017, Abruf: 06.06.2017

[Bia14] BIANCO, David: *The Pyramid of Pain.* http://detect-respond.blogspot.de/2013/03/the-pyramid-of-pain.html. Version: 2014, Abruf: 05.07.2017

[BZNT11] BURGUERA, I. ; ZURUTUZA, U. ; NADJM-TEHRANI, S.: Crowdroid: Behavior-Based Malware Detection System for Android. (2011)

[Chi13] CHICKOWSKI, E.: Top 15 Indicators Of Compromise. (2013). http://www.darkreading.com/attacks-breaches/top-15-indicators-of-compromise/d/d-id/1140647

[DV16] DAMRI, G. ; VIDYARTHI, D.: Automated dynamic malware analysis techniques for Linux environment. (2016)

[Eik17] EIKENBERG, R.: BSI warnt vor gefährdeten Cloud-Servern: Über 20.000 deutsche ownCloud- und Nextcloud-Installationen veraltet. (2017). https://www.heise.de/newsticker/meldung/BSI-warnt-vor-gefaehrdeten-Cloud-Servern-Ueber-20-000-deutsche-ownCloud-html, Abruf: 05.05.2017

[ese14] ESET: *Operation Windigo.* https://www.welivesecurity.com/wp-content/uploads/2014/03/operation_windigo.pdf. Version: 2014, Abruf: 26.07.2017

[fir17] FIREEYE: *Services.* https://www.fireeye.de/services.html. Version: 2017, Abruf: 11.08.2017

[GDA17] GDATA: *Was ist eigentlich ein Exploit?* https://www.gdata.de/

ratgeber/was-ist-eigentlich-ein-exploit. Version: 2017, Abruf: 11.08.2017

[Hum17] HUMMERT, Christian: Malware Forensics. In: LABUDDE, Dirk (Hrsg.) ; SPRANGER, Michael (Hrsg.): *Forensik in der digitalen Welt: Moderne Methoden der forensischen Fallarbeit in der digitalen und digitalisierten realen Welt*. Springer Berlin Heidelberg, 2017, Kapitel 7

[KB16] KÜHL, E. ; BREITEGGER, B.: *DDoS-Angriffe. Der Angriff, der aus dem Kühlschrank kam.* http://www.zeit.de/digital/internet/2016-10/ddos-attacke-dyn-internet-der-dinge-us-wahl. Version: 2016, Abruf: 21.03.2017

[KSK17] KHALIMONENKO, A. ; STROHSCHNEIDER, J. ; KUPREEV, O.: DDoS attacks in Q4 2016. (2017). https://securelist.com/analysis/quarterly-malware-reports/77412/ddos-attacks-in-q4-2016/, Abruf: 03.04.2017

[LAHR11] LIGH, Michael H. ; ADAIRE, Steven ; HARTSTEIN, Blake ; RICHARD, Matthew: *Malware Analyst's Cookbook and DVD. Tools and techniques for figthing malicious code.* Wiley Publishing, 2011

[LCLW14] LIGH, Michael H. ; CASE, Andrew ; LEVY, Jamie ; WALTERS, Aaron: *The Art of Memory Forensics: Detecting Malware and Threats in Windows, Linux, and Mac Memory*. Wiley Publishing, 2014

[Lee13] LEE, R.: *SANS DFIR. Digital Forensics & Incident Response. Finding Unknown Malware – Step-By-Step*. https://digital-forensics.sans.org/media/poster_fall_2013_forensics_final.pdf. Version: 2013, Abruf: 21.03.2017

[LK17] LIPOVSKY, Robert ; KÁLNAI, Peter: *KillDisk now targeting Linux: Demands $250K ransom, but can't decrypt*. https://www.welivesecurity.com/2017/01/05/killdisk-now-targeting-linux-demands-250k-ransom-cant-decrypt/. Version: 2017, Abruf: 10.07.2017

[LYW+16] LIAO, X. ; YUAN, Kan ; WANG, XiaoFeng ; LI, Zhou ; XING, Luyi ; BEYAH, Raheem: Acing the IOC Game: Toward Automatic Discovery and Analysis of Open-Source Cyber Threat Intelligence. (2016)

[MKK07] MOSER, Andreas ; KRUEGEL, Christopher ; KIRDA, Engin: Limits of Static Analysis for Malware Detection. (2007)

[nep16] NEPALISUPPORT: *Linux File System Hierarchy*. https://nepalisupport.wordpress.com/2016/06/29/linux-file-system-hierarchy/. Version: 2016, Abruf: 10.08.2017

[PL16] PILKINGTON, Mike ; LEE, Rob: *Know Abnormal...Find Evil*. https://digital-forensics.sans.org/media/poster_fall_2013_forensics_final.pdf. Version: 2016, Abruf: 26.07.2017

[Pol16] POLLAK, Ernst: *Einführung: Das Betriebssystem Linux*. http://www.ernstlx.com/linux90linux.html. Version: 2016, Abruf: 17.07.2017

[Qua12] QUADE, Jürgen: *Kernel-Rootkits und Gegenmaßnahmen Spitzel im Inneren*. http://www.linux-magazin.de/Ausgaben/2012/10/Rootkits. Version: 2012, Abruf: 08.06.2017

[Sch10] SCHMIDT, Jürgen: *Das Antivirus-Lexikon: Was bedeutet eigentlich....* https://www.heise.de/security/artikel/Das-Antivirus-Lexikon-Was-bedeutet-eigentlich-1105792.html. Version: 2010, Abruf: 17.08.2017

[Sel15] SELINGER, Markus: *Linux: 16 Schutzpakete gegen Windows- und Linux-Schädlinge im Test*. https://www.av-test.org/de/news/news-single-view/linux-16-schutzpakete-gegen-windows-und-linux-schaedlinge-im-test/. Version: 2015, Abruf: 07.08.2017

[SF12] SHAZAD, F. ; FAROOQ, M.: ELF-Miner: Using Structural Knowledge and Data Mining Methods To Detect New (Linux) Malicious Executables. (2012)

[SH12] SIKORSKI, Michael ; HONIG, Andrew: *Practical Malware Analysis. The Hands-On Guide to Dissecting Malicious Software*. no starch press, 2012

[SS17] SEARCHSECURITY ; SYNGRESS: *Malware Forensics Field Guide for Linux Systems: Digital Forensics Field Guides*. http://searchsecurity.techtarget.com/feature/Malware-Forensics-Field-Guide-for-Linux-Systems. Version: 2017, Abruf: 12.06.2017

[Tho16] THOMIAHONEN CONSULTING: *Anzahl der in Gebrauch befindlichen Smartphones weltweit nach Betriebssystem im März 2016 (in Millionen)*. https://de.statista.com/statistik/daten/studie/246004/umfrage/weltweiter-bestand-an-smartphones-nach-betriebssystem/. Version: 2016, Abruf: 09.05.2017

[Tri17] TRINKWALDER, Andrea: *Malware auf Zerstörungsjagd: BrickerBot legt unsichere IoT-Geräte still*. https://www.heise.de/newsticker/meldung/Malware-auf-Zerstoerungsjagd-BrickerBot-legt-unsichere-IoT-Geraete-still-367.html. Version: 2017, Abruf: 12.06.2017

[ubu15] UBUNTU: *RootSudo*. https://help.ubuntu.com/community/RootSudo. Version: 2015, Abruf: 07.06.2017

[w3t17] W3TECH: *Usage of operating systems for websites*. https://w3techs.com/technologies/details/os-linux/all/all. Version: 2017, Abruf: 16.05.2017

```
#!/bin/bash

#Skript zum Sammeln der benötigten Daten
#muss mit Root-Rechten ausgeführt werden

#lege Ordner an für Ergebnisse
mkdir -p results

#RAM Dump anlegen mit LiME
echo "lege RAM Dump an..."
make -C $PWD/LiME-master/src
kernelmodul=$(ls $PWD/LiME-master/src| grep .ko)
insmod $PWD/LiME-master/src/$kernelmodul "path=$PWD/results/ram.lime format=lime"
echo "RAM Dump erstellt"

# sammeln der /etc/passwd und /etc/shadow und /etc/group
echo "Sammle Userdaten"
cp /etc/passwd /etc/shadow /etc/group $PWD/results

# Start Sophos Scan und Rkhunter, wenn vorher auf System installiert
echo "Starte Malware Scan"
savscan / >$PWD/results/sophos.log
rkhunter -c
cp /var/log/rkhunter.log $PWD/results

echo "Skript beendet."
```

Abbildung .1: Quellcode des Skriptes für die Datenerhebung (Quelle: eigene Darstellung)

```
#!/bin/bash
# Script für die Analyse
# RAM Image wird Script Übergeben
# RAM Image liegt unter ./results/ram.lime
# Script mit Pfad zum RAM Image aufrufen

#Pfad zum RAM Image als Eingabe hinter Scriptaufruf
path=$1

#Profilname für Volatility
profilname="LinuxUbuntu-17_04-serverx64"

# Volatility starten
python ./tools/volatility/vol.py -f $path --profile=$profilname linux_check_syscall >check_syscall
python ./tools/volatility/vol.py -f $path --profile=$profilname linux_check_modules >check_modules
python ./tools/volatility/vol.py -f $path --profile=$profilname linux_check_inline_kernel >check_inline_kernel
python ./tools/volatility/vol.py -f $path --profile=$profilname linux_psaux >pslist
python ./tools/volatility/vol.py -f $path --profile=$profilname linux_pstree >pstree
python ./tools/volatility/vol.py -f $path --profile=$profilname linux_pidhashtable>pidhashtable
python ./tools/volatility/vol.py -f $path --profile=$profilname linux_lsmod >lsmod
python ./tools/volatility/vol.py -f $path --profile=$profilname linux_mount >mount

# Verarbeitung der wichtigsten Daten
awk '{print $2}' ./results/pidhashtable >unsort_pidhashtable
awk '{print $2}' ./results/pslist >unsort_pslist

sort unsort_pidhashtable >sort_pidhashtable
sort unsort_pslist >sort_pslist

# Vergleicht die erstellten Prozesslisten miteinander
diff sort_pslist sort_pidhashtable >prozess_vergleich

# Vergleicht die erstellte User-Liste mit der Whitelist
diff passwd ref_passwd > user_vergleich

# Sucht nach manipulierten Systemcalls
cat check_syscall|grep HOOKED >hooked_syscalls
```

Abbildung .2: Quellcode des Analyse-Skriptes (Quelle: eigene Darstellung)

Erklärung

Hiermit erkläre ich, dass ich meine Arbeit selbstständig verfasst, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt und die Arbeit noch nicht anderweitig für Prüfungszwecke vorgelegt habe.

Stellen, die wörtlich oder sinngemäß aus Quellen entnommen wurden, sind als solche kenntlich gemacht.

Mittweida, 24.08.2017